Netherlands
organization for
applied scientific
research

TNO-report

FEL

TNO Physics and Electronics
Laboratory

P.O. Box 96864
2509 JG The Hague
Oude Waalsdorperweg 63
The Hague. The Netherlands

Fax +31 70 328 09 61
Phone +31 70 326 42 21

TD 93-0055

(1)

report no.
FEL-92-B059

copy no.
2

title

Intelligent tutoring systems; report of one year sabbatical leave in the USA

author(s):

J.G.M. van der Arend

date:

February 1992

classification	
title	: unclassified
abstract	: unclassified
report text	: unclassified
appendices A-I	: unclassified

no. of copies	: 29
no. of pages	: 146 (incl. appendices, excl. RDP + distr. list)
appendices	: 9

TNO

report no. : FEL-92-B059

title : Intelligent tutoring systems; report of one year sabbatical leave in the USA

author(s) : J.G.A. van der Arend

institute : TNO Physics and Electronics Laboratory

date : February 1992

NDRO no. :

no. in pow '92 : 706

Research supervised by: H. Kuiper

Research carried out by : J.G.M. van der Arend

ABSTRACT (UNCLASSIFIED)

This report gives an overview of my sabbatical year in the USA from September 1990 until September 1991. For almost four months, I worked on the ISTS project at the Embry-Riddle Aeronautical University (ERAU) in Daytona Beach, Florida. The ISTS (Intelligent Simulation Training System) is a training system for the air traffic controller using artificial intelligence techniques. I have developed a limited prototype of the ISTS on the Sun SPARCstation using the simulation package MODSIM II.5.

The second part of the study year was at the U.S. Army Research Institute for Behavioral and Social Sciences (ARI) in Alexandria, Virginia. I worked on the analysis and design of Organon, a tool for graphical knowledge organisation. Organon is the foundation for a future instructional design environment. I built a prototype of Organon on a NeXT workstation using the object-oriented prototyping tool Interface Builder and the programming language C.

The sabbatical year was very useful and instructive. Some of the information, knowledge and skills I already used in projects at TNO/FEL. The field of instructional design is very appropriate for TNO/FEL because in the future many research projects will be in this field. The Organon concept can also be used in the future authoring environment of the FELIX intelligent tutoring system.

rapport no.	:	FEL-92-B059
titel	:	Intelligente onderwijssystemen; verslag van één jaar uitzending naar de VS
auteur(s)	:	Ir. J.G.M. van der Arend
instituut	:	Fysisch en Elektronisch Laboratorium TNO
datum	:	februari 1992
hdo-opdr.no.	:	
no. in iwp '92	:	706
Onderzoek uitgevoerd o.l.v.	:	Ir. H. Kuiper
Onderzoek uitgevoerd door	:	Ir. J.G.M. van der Arend

SAMENVATTING (ONGERUBRICEERD)

Dit rapport geeft een overzicht van mijn werk in het kader van de uitzending naar de Verenigde Staten van september 1990 tot september 1991. In Daytona Beach, Florida heb ik vier maanden gewerkt aan het ISTS project op de Embry-Riddle Aeronautical University (ERAU). Het ISTS (Intelligent Simulation Training System) is een trainingssysteem voor luchtverkeersleiders gebruikmakend van kunstmatige intelligentie technieken. Ik heb een beperkt prototype van het ISTS op een Sun SPARCstation ontwikkeld. MODSIM II.5 is het gebruikte simulatie-pakket.
Het tweede deel van het jaar speelde zich af op de U.S. Army Research Institute for the Behavioral and Social Sciences (ARI) te Alexandria, Virginia. Ik heb daar gewerkt aan de analyse en ontwerp van Organon, een gereedschap voor grafische kennisorganisatie. Organon is bedoeld als basis voor een toekomstige opleidingsontwikkelomgeving. Ik heb een prototype van Organon op een NeXT computer gebouwd, gebruikmakend van het object-georienteerde gereedschap om prototypen te maken (Interface Builder) en de programmeertaal C.

Het studiejaar was erg leerzaam en de verworven kennis en vaardigheden zijn goed toepasbaar in projecten op het FEL-TNO. Het vakgebied van opleidingsontwikkeling is eveneens erg toepasselijk voor het FEL-TNO omdat in de toekomst vele onderzoeksprojecten in dit vakgebied zullen worden uitgevoerd. Het Organon concept zou in de toekomst ook gebruikt kunnen worden voor een auteursomgeving van het intelligente onderwijssysteem FELIX.

CONTENTS

INTRODUCTION

This document is a report of the work I performed at a university and a research institute in the USA from October 1990 until September 1991.
From October until January, for almost four months, I worked on the ISTS project at the Embry-Riddle Aeronautical University (ERAU) in Daytona Beach, Florida. The ISTS (Intelligent Simulation Training System) project is a training system for the air traffic controller. I developed a limited prototype of the ISTS on the Sun SPARCstation using the simulation package MODSIM II.5. Dr. Andrew J. Kornecki, associate professor of the Aviation Computer Science Department and technical project leader of the ISTS project at ERAU, was my mentor. My work at ERAU is described in chapter 1.
For the second part of the year I had to move north to the U.S. Army Research Institute for Behavioral and Social Sciences (ARI) in Alexandria, Virginia. I stayed 8 month in Alexandria, from the end of January until the end of September, to work on the analysis and design of the Organon system and I built a prototype on a NeXT workstation using the object-oriented prototyping tool Interface Builder and the programming language C. Organon, a tool for graphical knowledge organiation, is the foundation for a future instructional design environment. Person of contact over there was Dr. Robert J. Seidel PhD, chief of Automated Instructional Systems Technical Area. I worked together with Dr. Larry W. Brooks in the Automated Training Development Research Team of Dr. Ray S. Perez. Chapter 2 gives an overview of my work at ARI.
Chapter 3 gives some conclusions of my study year.

Most of the appendices are independent reports, made during the year. Some are included without changes, others are summarized.

1 EMBRY-RIDDLE AERONAUTICAL UNIVERSITY

Since 1926, Embry-Riddle Aeronautical University provides men and women with an education in the aviation and aerospace fields. At Embry-Riddle Aeronautical University (ERAU) aviation is the total focus, not just a department or centre. Embry-Riddle has two residential campuses in Daytona Beach, Florida and Prescott, Arizona. Approximately 7,000 undergraduate students are enroled at the residential campuses and another 3,000 students are enroled in graduate programs. Student come from all 50 states and various foreign countries.

I worked at the Daytona Beach campus. This campus contains 22 buildings, which are located adjacent to the Daytona Beach Regional Airport and within an hour from Orlando and Kennedy Space Center at Cape Canaveral.

The library houses books, periodicals, documents, newspapers and media programs related to aviation. The Aviation Technology Center houses classrooms, simulators, weather room and dispatch headquarters. Here, the flight instruction is given in the Embry-Riddle fleet of over hundred aircraft. The Aviation Maintenance Technology Center is the place for instruction in maintenance and repair of fixed-wing and helicopter airframes and powerplants. The Engineering Science Laboratories building houses wind tunnels and a smoke tunnel, structures, materials, aircraft design and composite materials laboratories as well as a CAD system. The Airway Science Simulation Laboratory (ASSL) simulates the various elements of the national airspace system and is a center for aviation research and education. The elements include air traffic control, pilot simulators, weather information, airports, airways, pilot and aircraft performance. The Aviation Computer Science department provides several classrooms with machines like: Sun workstations for programming tasks, PCs for word processing and programming, PCs for various computer-based instruction, PCs for air traffic control simulation, flight simulators.

Dr. Andrew J. Kornecki is associate professor of computer science of the Aviation Computer Science (ACS) department. Department Chair is Dr. Iraj Hirmanpour, who is also the principal investigator for the Intelligent Simulation Training System (ISTS) project. I came to the ERAU university to work on this project. Dr. Kornecki works part-time on the ISTS project as the technical project leader. He is a specialist in computer simulations. Besides participation in several other research projects, he teaches basic and advanced computer science principles and techniques. The ASSL is the place where the Intelligent Simulation Training System project takes place.

1.1 The ISTS project

The Intelligent Simulation Training System (ISTS) is a collaborative research effort of the Embry-Riddle Aeronautical University with the University of Central Florida (UCF) and the General Electric Company (GE). The objective is to utilize artificial intelligence technology and simulation technology to create a learning/research environment in which cognitive, problem solving and judgement skills can be developed. Currently, the air traffic control (ATC) is the domain of the training system, but application of the technology is more far-reaching.

At the moment the outcome of the ISTS project is the prototype of the ISTS concept running on a Symbolics LISP computer and the ATC expert system with almost hundred rules. This expert system is made with the VP Expert expert system shell on IBM-compatible PC. When I arrived at the ERAU the project was in the process of finishing the Symbolics prototype and making attempts to convert the prototype to a Sun workstation. They were in the phase were they had to decide which programming language and which expert system shell to use.
The concept of the Intelligent Simulation Training System was originated in 1986 at the University of Central Florida (UCF) at Orlando. The project to proof the concept is a cooperative effort of UCF, GE and ERAU. GE made the simulation component, ERAU provides the ATC knowledge bases and UCF is responsible for the tutoring components and global system integration. The ISTS team at ERAU, including Dr. A. Kornecki, Dr. M. Towhidnejad, V. Galotti (former air traffic controller) and 4 students, held their project meeting every week. Meetings with the other ISTS partners (UCF and GE) were irregular, two or three times a month.

Air traffic control (ATC) is the service to provide a safe and orderly flow of air traffic. The US airspace is divided into 24 areas and aircraft separation responsibility within each area have been assigned to the air route traffic control center (ARTCC). To accomplish this task an area is divided into sectors. An air traffic controller is responsible for a sector. The supporting hardware in about 400 fully equipped airport towers, includes numerous computers along with communication, navigation, radar, ground and airborne equipment. In this environment with some 15,000 airport facilities, 400,000 miles of airways, 850,000 pilots, weather stations, military, naval and space operations the air traffic controller forms an important factor. The controller has a complex task interacting with the ATC System, with pilots and other controllers. The normal education and training time is at least three years before reaching the full performance level.

In essence, the functions of the controller are:

- Coordinating with other sectors
- Navigating aircraft
- Issuing altitude clearances
- Maintaining aircraft separation

In the controller's actions, the following activities are distinguished: initial clearance; departure; conflict separation; hand-offs; arrival; pilot requests and emergencies; and weather and traffic advisories.

1.1.1 The Symbolics prototype

The global architecture of the Intelligent Simulation Training System (ISTS) is as described in figure 1.1.1.

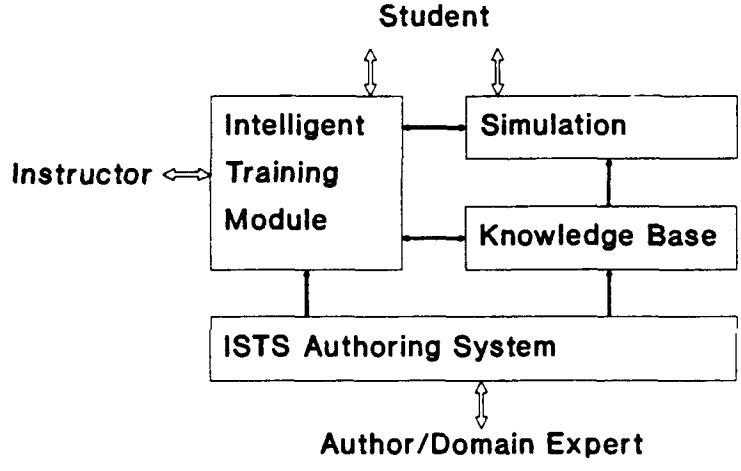

Figure 1.1.1 ISTS architecture

The ISTS is functionally divided into 14 modules: Expert, Expert Instructor, Tutor, Student Model, Evaluator, Diagnoser, Author, Discourse, Simulation, Input Filter, Intelligent Preprocessor, Translator, Control and Inference Engine.

The whole ISTS distinguishes three user categories: student, instructor and author user. The instructor can start and end a student session and can view the student records. The author user enters the domain-related knowledge.

The domain knowledge, stored as rules and facts, include navigating aircraft and maintaining aircraft separation. Other controller's functions, e.g. coordinating with other sectors and issuing altitude clearances, are not covered in the training system. The trainer can operate in five modes: reference source, demonstrator, coach, monitor and grader.

The simulation system simulates the functionality of the radar console control, the radar display and the movement of the aircraft. The system can operate in a stand alone mode separate from the rest of the ISTS. Further the student has a discourse monitor, which handles all non-simulation communication of student and instructor with the ISTS.
So, the student has two monitors: simulation and discourse monitor. The simulation monitor displays the radar display, the radar console controls and the movement of the aircraft. These controls include buttons, knobs and a trackball which are all accessible to the user through the mouse. A schematic impression of the screen layout is presented in figure 1.1.2.

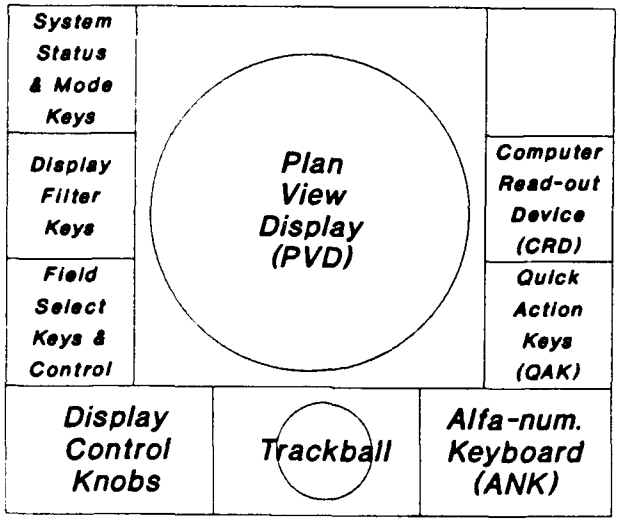

Figure 1.1.2 Radar control and displays

The discourse monitor provides an interface for all non-simulation communications. The screen is divided in windows for user input and text presentation.
The student can ask questions, enter comments and request for tutoring mode. The questions reflect the student's need to understand the system or a lesson. Typical questions are: 'What action

is recommended?', 'Why is an action recommended?' and "What are the alternatives?'. The student's comments are saved for later review by the instructor.
The instructor user can input through a menu the objective of the lesson, level of difficulty etc., based on the student's progress.

The ISTS system is developed on a Symbolics 3765 LISP machine. The programming language is LISP and the expert system shell ART (Automated Reasoning Tool).
They are currently working on an ISTS implementation on Sun SPARCstation in combination with the programming language C.
The system concept should work for a variety of domains. Ideas about implementing car driving lessons have not been implemented yet.

1.1.2 The Sun prototype

The Intelligent Simulation Training System (ISTS) project is working towards an implementation of the system on the Sun SPARCstation. In this light, I made a prototype of the ISTS with the simulation package MODSIM II on the SPARC. This prototype showed the possibilities for interfacing with the user. The prototype also showed the capabilities of the SPARC and of the MODSIM package.

The package MODSIM II, presented as 'The language for object-oriented programming', contains the MODSIM II language (version 1.4) and the graphics and animation presentation tool SIMGRAPHICS II (version 1.1). The MODSIM language is a general-purpose, modular, block-structured high-level programming language which provides direct support for object-oriented programming and discrete-event simulation. The MODSIM language is similar to an object-oriented version of the Pascal programming language with extensions. It runs under Unix-like SunOS (release 4.1) in combination with the window environment SunView, but it is supported on a variety of machine architectures and operating systems. MODSIM II programs are portable from one machine to another.
An object in MODSIM II is a dynamic data structure that includes an associated list of fields or attributes (variables) and an associated list of methods or services (routines). The definition of an object type can be in terms of another, already-existing object (inheritance). When the object type is defined in terms of two, or more, already-existing object types, then the term multiple inheritance is used. One particular object of an object type is called an instance.

With the Sun ISTS prototype the user is able to practice the skills of the en-route air traffic controller. The user (or student) gets on the screen:

1 plan view display,
2 three flight strips bays,
3 window for the display of the incoming messages,
4 icon representing the communication to the pilot,
5 icon representing the communication to the other controllers,
6 simulation time.

These different elements are divided on screen in the following way:

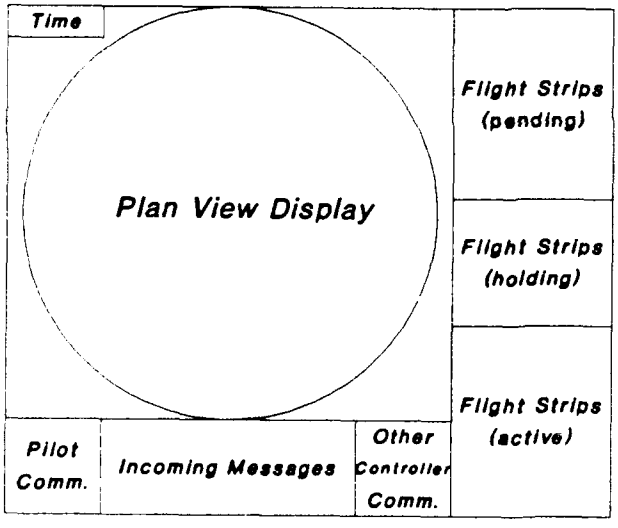

Figure 1.3 Sun ISTS prototype's screen layout

On the scope the user sees an radar image of the sector he/she has to control. In the current version is that the high altitude sector Lake City (FL 240 and above) from the Jacksonville ATC Centre in Florida. The description of the airspace around this sector (fixes, airways and airroutes) come from the Airspace file. The Airspace file is the data file with definition of the airspace sector data, fixes, airways and routes).

The aircraft are being generated from the Scenario file or randomly, according to the choice of the user. The Scenario file is the data file with definition of the scenario (global scenario data and aircraft descriptions).

The Sun prototype is able:

1 to let the user manipulate the flight strips, like:
   - update the contents of the strip;
   - moving a strip from a bay to another;
   - changing the sequence of the strips in a bay;
   - deleting a strip.
2 to let the user communicate with pilots, by constructing and sending messages to pilots and by displaying messages from pilots.
3 to simulate the communication from other controllers, by constructing and sending messages to other controllers and by displaying messages from other controllers.
4 to simulate the execution of controller actions, like: hand off procedure with other sectors.
5 to simulate the execution of aircraft actions, like:
   - climbing to a higher altitude;
   - descending to a lower altitude;
   - turning the aircraft to a specified heading (left or right);
   - flying to a specified fix;
   - holding the aircraft at the current position.

The Sun prototype gives a good impression of the capabilities of the Sun SPARC and the MODSIM package. Although the results are promising, the prototype still is limited. For example, there is no intelligence in the program and no analysis or judgement of the student's performance. Some other limitations were due to the MODSIM package, e.g. limited dialogue boxes and menu bars.

1.2 Results

After a few weeks of browsing through the ISTS reference materials, I started to write down my impressions about the ISTS project. These notes are gathered in the report 'Notes on my work at ERAU' (see appendix A). Once I got involved into the project and became familiar with the ISTS concept and prototype, I made a short comparison between the artificial intelligence of the ISTS system and the FELIX system of TNO/FEL. This comparison is described in the report 'The AI aspects of the ISTS' (see appendix B). The motivation to do the comparison was a result of my presentation of the FELIX system to the ISTS project members. After a month, at the end of

October, I started to play around on the Sun SPARCstation in the ASSL, mainly installing and learning to use the MODSIM package. The experiences with MODSIM are described in report 'MODSIM II' (see appendix C). The prototype I made with MODSIM, is described in report 'The ISTS prototype on the Sun SPARCstation' (see appendix D). To provide computer-based training using the Sun prototype, I made some suggestions to enhance this prototype in report 'CBT in the ISTS' (see appendix E).

During my work at the ERAU, I got the opportunity to visit the Jacksonville air traffic control center in Hilliard. It was very useful in understanding the training and work of an air traffic controller. They use simple CBT to train airport abbreviations and aircraft navigation. On the other hand, the training of a coming assistant controller takes five other persons (controller, instructor, pilots and other controllers) on a stand-alone ATC training system.
At the ERAU university, the NeXT computer was presented. The NeXT is a new computer and with OpenWindows capabilities and an excellent integration of the optical disk. The NeXT computer can run MS-DOS applications from MS-DOS diskettes and costs close to $10,000. They are currently making the NeXT able to execute Macintosh applications. During my work at the Army Research Institute I used the NeXT extensively; for more information about the NeXT the reader is referred to chapter 2.
Further, I attended a presentation of TAAM (Total Airspace and Airport Modeller). It is a fast and sophisticated tool for airspace and airport managers running on the Sun SPARC. It can simulate the flow of air traffic with a 3D-viewer. For example, the user can make visible the consequences of adding a departure lane or increased flow of traffic.
At the end of my stay in Daytona Beach I was invited to the Simulation and Control Systems Department of General Electric in Daytona Beach. Among others, they showed me the COMPU-SCENE PT2000, a visionics system for armor training, low range low-level flight training and hardware-in-the-loop sensor simulation. The first application of this system is the prototype M-1 Tank Driver Trainer developed for the U.S. Army.

After I left Daytona Beach, Dr. Kornecki and I wrote a paper, titled 'Object-oriented Simulation of an En-route ATC Sector', which is submitted to the SCS Simulation journal.

2 U.S. ARMY RESEARCH INSTITUTE

The U.S. Army Research Institute for the Behavioral and Social Sciences is the research organisation of the U.S. Army. The headquarter and research center of the Army Research Institute (ARI), technical director Dr. Edgar M. Johnson, is situated in Alexandria, Virginia with 13 field units and scientific coordination offices throughout the United States and Europe. Over 300 scientists, technicians, military professionals and support staff are working for the ARI. In Alexandria the ARI is located on the 6th floor of the AMC (Army Materiel Command) building.

The ARI incorporates 3 research laboratories, i.e. the Manpower & Personnel Research Laboratory, the Systems Research Laboratory, and the Training Research Laboratory, and an Office of Basic Research. The Training Research Laboratory includes several teams, i.a. Leadership and Motivation Technical Area, Aviation R&D Activity, Ft. Knox Field Unit, Ft. Benning Field Unit and Automated Instructional Systems Technical Area. The Automated Instructional Systems Technical Area (AISTA) team embodies around 25 persons and is lead by Dr. Robert J. Seidel. The mission of Seidel's team is to increase the effectiveness and efficiency in the development and delivery of training by application of state-of-the-art computing technologies. Projects are related to computer-based training development systems and AI-based tutors. The teams within AISTA are:

- Automated Training Development Research Team (Dr. R.S. Perez),
- Smart Technology for Language Learning Team (Dr. J. Psotka),
- Applications of Intelligent Training Systems Team (Dr. A. Mirabella) and
- Learning Technology Team (Dr. R. Wisher).

The team of Dr. Seidel consists of research psychologists with little of no technical background. Example projects within the AISTA team are research into: the skill acquisition and retention by using the Morse code domain (Bob Wisher) and the acquisition and retention of foreign language skills (Melissa Holland).

The Automated Training Development Research Team, lead by Dr. Ray S. Perez, develops and evaluates training development job aids or tools. Their goal is to develop technologies to make the training development process more effective and efficient. Within this team four ARI scientists (Ray Perez, Ok-choon Park, Larry Brooks, Douglas MacPherson) are working together with approximately four students, mainly from the George Mason University. I was working in this team, which was involved in the development of important instructional methods and

systems, e.g. ASAT (automated system approach to training, AKAT (automated knowledge acquisition tool), ID Expert (instructional design expert system), and KA/IAA system (knowledge acquisition/ intelligent authoring aids).

At that moment, Ray Perez was working on a project to build a cognitive model of an instructional developer. The project was in the initial phases of interviewing instructional developer experts. Ok-choon Park was working on the investigation of the characteristics of mental models and training strategies for formation of mental models. A mental model is a concept to explain human mental processes of understanding and translating external reality into internal representation. Larry Brooks was developing a knowledge acquisition computer tool, called Organon. During my stay at ARI, I primarily worked on the second version of Organon.

2.1 Organon

This section presents the system Organon that will be able to assist humans in organizing and analyzing complex bodies of information for the purpose of solving knowledge rich and ill defined problems such as the construction of lessons for computer-assisted instruction (CAI) and the development of knowledge-based models for intelligent tutoring systems (ITS). Succinctly, Organon is a software tool that lets users enter and organize information using a graphical interface built around the notion of nodes and links. A previous, prototype version of Organon exists (Brooks, in press; Brooks & Jardine, in press). However, in the course of developing the prototype it became obvious that a second version of the software would have to be built in order to meet the requirements of the project. These requirements are:

(1) The software would have to meet some of the basic needs for expert problem solvers such as the ability to chunk information, to view knowledge from multiple viewpoints, and to work on a problem in a top-down and a bottom-up manner;

(2) The software must serve as a foundation on which domain dependent knowledge organization tools could be built;

(3) The software must allow for the relatively easy extension of its basic capabilities so that future enhancements to the system will not require the underlying software to be re-written; and

(4) The software should be written so that it is oriented toward having a group of people work on organizing the same knowledge base.

To some degree all of these requirements have been met by the analysis and its descendant Organon version 2 (van der Arend and Brooks, in press). The first requirement is met by keeping and expanding the features present in the first Organon prototype. Enhancements to the prototype include: the use of domains as a high level knowledge organization technique; the use of virtual nodes to allow for complex information webs to be built using a simple user interface, and additional changes that increase the utility of the software for use by a group. Requirements 2 and 3 are primarily met by using an object-oriented approach to analyzing Organon. The use of object-oriented analysis and design facilitates both the expansion and reuse of existing (once it is created for the new version). The last requirement on providing more group oriented features serves more as a pointer to future development than it does as a criteria for the existing analysis. Nevertheless, this requirement was addressed in part by explicitly having the users of the system identify themselves and the other members of the team working on a particular project.

In order to give the reader the appropriate background for understanding the software analysis, in the next two sections, I will: briefly review the existing Organon prototype (section 2.1.1), and present an introduction to the object-oriented analysis method that we used (section 2.1.2). After that I will discuss the Organon analysis with an emphasis on the classes and objects that are defined for the system. Next, I will continue the analysis by presenting the design and subsequently the limited prototype of version 2.

2.1.1 Organon version 1

In this section, a short overview of the Organon version 1 prototype, that served as the basis for the current analysis, is presented. This overview is not intended to be complete, but to give the reader a feel for what Organon version 2 would be like. The existing prototype for Organon version 1 is implemented in Common Lisp on the Macintosh II family of computers. Originally, the Organon project began as an attempt to create a knowledge engineering environment called KEEN (Knowledge Engineer's Electronic Notebook). As the project progressed, it became apparent that a more fundamental tool for organizing knowledge or information for a variety of purposes should be constructed. This led to a change in emphasis during the development of the prototype and directly influenced our thinking in doing the development of the version 2 prototype.

The Organon version 1 prototype allows users to construct a graphical network of nodes and links where the nodes can represent concepts, facts, rules, etc., and the links represent named

associations between the nodes. A node is represented on screen by a rectangle, the node's box. The box is labelled by the name of the node. Links are represented on screen by lines with arrows. By entering nodes and links, a network of boxes (the nodes) and interconnecting lines (the links) arise.

Each node contains slots or attributes pertaining to the node's name, type, and description. These slots are user accessible and can be modified at any time by the user. Additionally, there are slots for each node that are hidden from the user and are used by the system. These slots contain information concerning the placement of the node's box, the other nodes to which the node is linked, the parent and child nodes, etc.

The working of Organon version 1 is graphically as far as possible. To create a new node the user double clicks in the Organon workspace window with the mouse. A pop-up dialogue window appears immediately after the double click. This window asks the user to provide a name, type, and description for the node. The new node is then placed on the window at a location where the mouse cursor was at the time of the double click.

The creation of a new link is similar. A new link is made when the user double clicks the mouse button while the mouse cursor is inside an existing node. Then, continuing to hold down on the mouse button after the second click, the user moves the mouse cursor to another node. A line is drawn from the first to the mouse cursor while this takes place. Next, a pop-up window appears on the screen and requesting the user to enter information concerning the type of link to be created. If the type is not known, then a request is made to the user to provide a text description of this type of link. Last, if the type of node is new to the system, then the type is added to the links menu in the top menu bar.

Finally, to move and relocate nodes on the screen, the user clicks the mouse button once while the cursor is inside a node, and then holding the mouse button down, moves that node to the desired position. When the mouse button is released, the node is re-drawn in the position occupied by the ghost box, and all connected links are re-drawn to the moved node.

The prototype allows for a greater range of action than those just described, but these actions are the ones that are central to understanding how the software to be built using the analysis presented here should work. For a more detailed description of the prototype, the reader is referred to Brooks (in press).

2.1.2 Object-oriented approach

For the development of Organon version 2, Larry and I used the object-oriented approach. The general notion underlying the object-oriented approach is that software can be thought of and developed from a data perspective as opposed to a procedure perspective. That is, instead of thinking of a computer program as a collection of procedures that are called on to operate on various pieces of data, one can think of a program as a collection pieces of data that are called on to operate on themselves. Some of the most important characteristics of the object-oriented approach are:

(1) A data module or unit is called an object, and objects have associated functions (sometimes called generic functions) that know how to perform operations on a certain object. Objects are typically thought of as data structures that have slots that contain values. These values can be read or changed by an object's associated functions.

(2) Objects belong to classes. A class is an abstract data type that can be thought of as a template for a group of objects that captures the commonalties among those objects. Objects that belong to a class are often called instances of that class.

(3) Classes are structured in a hierarchical manner, and characteristics of higher level classes can be inherited by lower level classes. These inherited characteristics include slots, slot values, and functions.

(4) Objects communicate by passing messages to themselves and to other objects. Messages typically ask an object to perform some action on itself via an associated function. Messages are also called methods.

(5) How an object is structured and how functions work on that object is invisible to other objects. In other words, there is no need to know the internal workings of an object in order to use it. This is called encapsulation.

Object-oriented analysis (OOA) is a way of analyzing a problem domain taking advantage of the benefits of an object-oriented approach. In this respect OOA is similar to older software analysis techniques that have been developed for other styles of programming such as structured programming. Some examples of these methods are functional decomposition, data flow and information modeling. OOA builds upon each of these techniques. From information modeling come constructs analogous to attributes, instance connections, generalization-specialization, and whole-part. From OOPL and KBS come the encapsulation of attributes and exclusive services, communication with messages, generalization-specialization, and inheritance.

The particular OOA method that we used in our analysis is the one suggested by Coad and Yourdon (1991). Larry and I choose this method because of its simplicity and language independence. From this point on, when we use terms associated with OOA, we specifically mean the version of OOA prescribed by Coad and Yourdon. The Coad and Yourdon model consists of five layers (see Figure 2.1.1):

---------- Subject layer ----------
---------- Class-&-object layer ----------
---------- Structure layer ----------
---------- Attribute layer ----------
---------- Service layer ----------

Figure 2.1.1 The OOA multi-layer model

The five major activities of OOA are: (1) Finding class-&-objects, (2) Identifying structures, (3) Identifying subjects, (4) Defining attributes, and (5) Defining services. It is helpful to move from one activity to the next, but it's not mandatory. The analyst may do the activities in the sequence he or she prefers. The strategy and contents of this analysis is extensively described by Coad and Yourdon (1991).

The OOA activities form the corresponding layers of the OOA model. The model includes its notation symbols and the class-&-object specifications.

Object-oriented design is the next step in the object-oriented development. In essence this step is easy to explain. The OOA model is the Problem Domain Component, embedded into the OOD model (see figure 2.1.2).

Human Interaction	***Problem Domain***	***Task Management***	***Data Management***
------Subject------	------Subject------	------Subject------	------Subject------
--Class-&-obj--	--Class-&-obj--	--Class-&-obj--	--Class-&-obj--
----Structure----	----Structure----	----Structure----	----Structure----
----Attribute----	Attribute----	----Attribute----	----Attribute----
---Service---	---Service---	---Service---	---Service---

Figure 2.1.2 The OOD multi-layer model

The OOD model is a multi-layer model with the following components: Human Interaction Component, Problem Domain Component, Task Management Component and Data Management Component

2.1.3 Organon version 2

This section describes the object-oriented analysis and design of Organon version 2. As a part of the design phase, we developed a prototype of version 2 on the NeXT computer.

The object-oriented approach led to a variety of object classes. The main class-&-objects, i.e. classes with object instances, are:

- Node
- Link
- Group
- Domain

A node is a piece of knowledge and is described by the node's name, description and version number. Additional node information, like the node type, the knowledge reference, and the author, can be specified. The node type categorizes the chunk of information. Possible types are: concept, process, activity, fact and procedure. The reference is a reference to the source of the knowledge. This might be a description of a book, person or other knowledge source. The author is a reference to the person who enters the knowledge.

A link is a connection between two nodes and is determined by begin node, end node and link type. A link is described by name, description and version number; additional information includes also link type, reference and author.

A group is a set of nodes. Usually, a group includes a network of nodes and interconnecting links. The group has the same attributes like the node: name, description, version number, type, reference and author.

Each node is placed in a hierarchical node tree. This means that each node has a parent node. Only the top node of the tree has no parent node. Seen from the other way, it is clear that a node can have one or more child nodes, but a node might be childless. The tree structure of nodes with interrelating links and groups is a domain. So, within a domain, the knowledge is chunked and organized into the domain units: nodes, links and groups. The various types, e.g. node, link and group types, to specify the domain units can be defined within the domain.

The previous will be illustrated by the next example domain of preparing and serving the dutch pea soup. A possible decomposition is presented in figure 2.1.3.

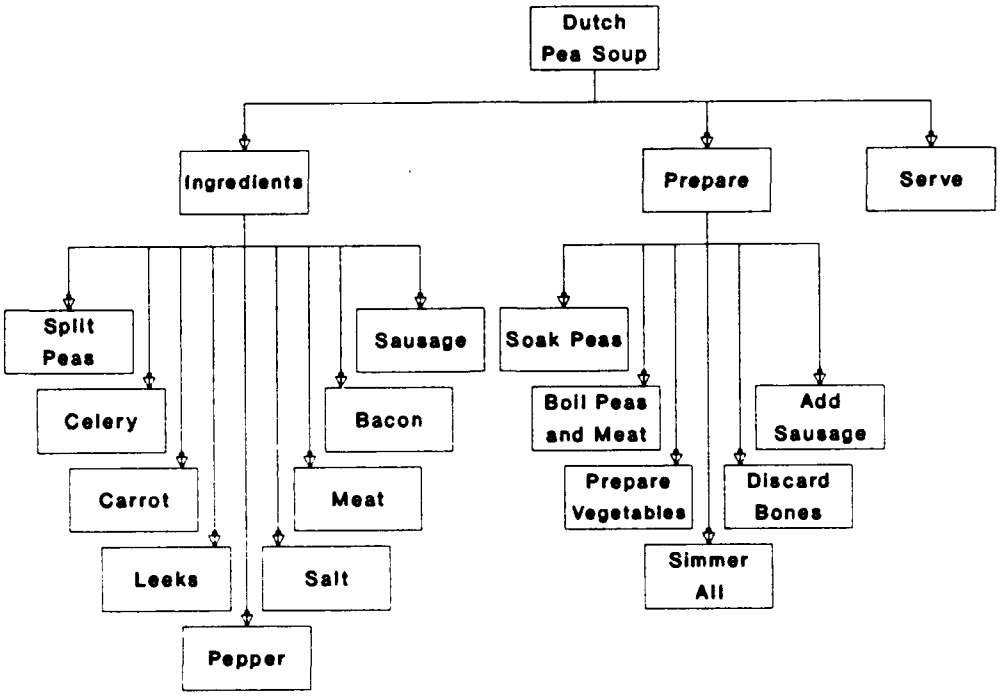

Figure 2.1.3 Example Organon Domain: Dutch Pea Soup.

The nodes are represented by circles, the links are represented by lines and arrows. In the example, the top node 'Dutch Pea Soup' has three child nodes: 'Ingredients', 'Prepare' and 'Serve'. The difference of the child nodes' types are not visible in figure 2.1.3; The node 'Ingredients' has node type 'Contents', the other two nodes have node type 'Activity'.

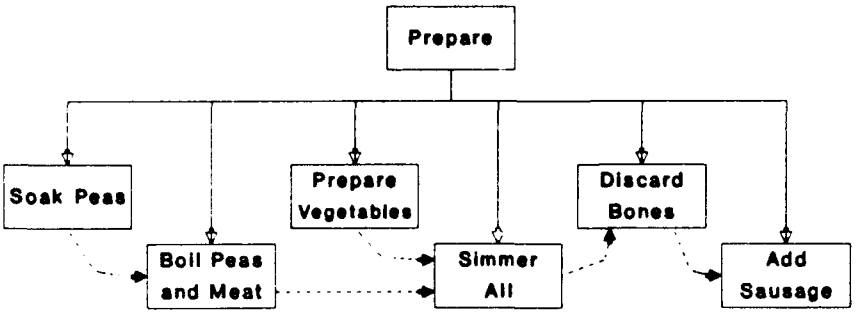

Figure 2.1.4 Example Organon Links.

Links between parent and child nodes, the so-called vertical links, have an open arrow head, and horizontal links have an closed, black arrowhead. In figure 2.1.4, horizontal links are shown. In

this example all links have the link type 'Conditional', meaning that the begin node's activity has to be finished before the end node's activity can be executed.
In figure 2.1.5, a part of the pea soup example is expanded by visualizing possible groups. The child nodes of the node 'Ingredients' can be grouped into groups 'Vegetables', 'Herbs and Spices' and 'Meat'.

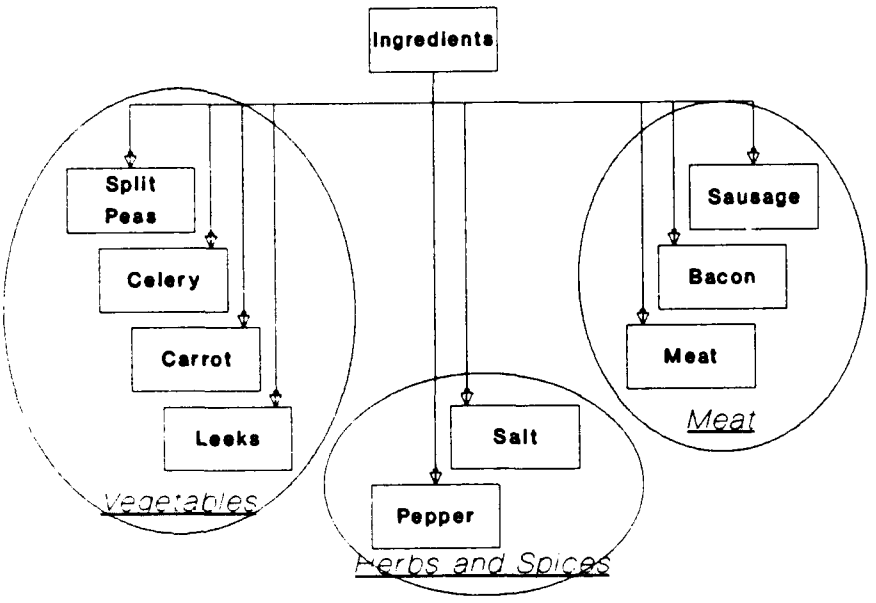

Figure 2.1.5 Example Organon Groups.

With Organon the user can organize knowledge to describe a specific domain knowledge. One of the strong points of Organon is that the domain knowledge can be seen in different ways and from various viewpoints. The nodes can be represented by box or text label. The structure and contents can be made visible by one node only, at one level or at various levels at the same time. Also, node types and link types can be made visible or invisible.

After a thorough analysis and design of the Organon version 2, it took two months to produce a limited prototype of Organon version 2 on the NeXT workstation. The NeXT workstation is powerful and user-friendly UNIX environment with tools for object-oriented system prototyping and development. One such a tool is the Interface Builder, which allows the programmer to graphically design an application interface and define the relation between objects. To develop the Organon prototype we used this Interface Builder and Objective-C.

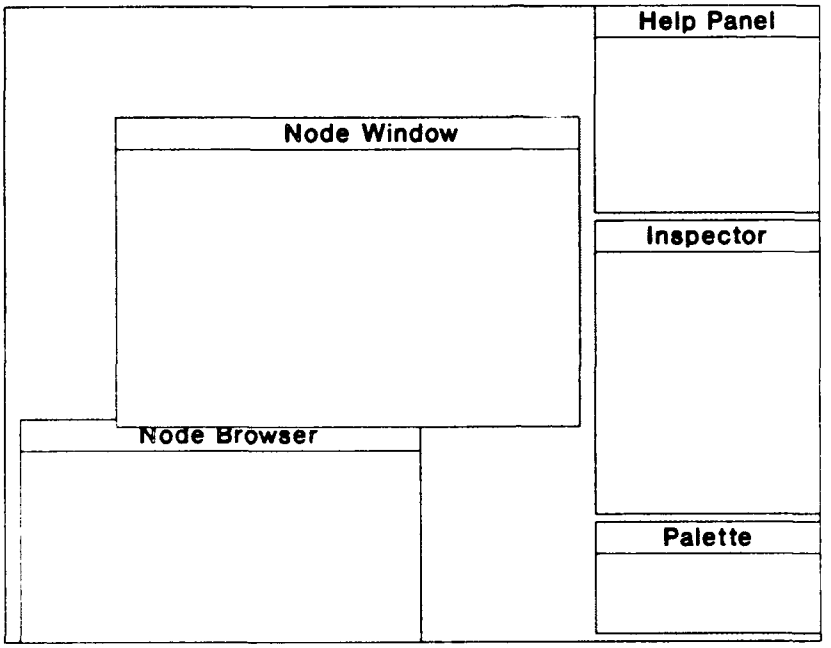

Figure 2.1.6 Organon's screen layout

Figure 2.1.6 shows screen layout of the Organon version 2 prototype.
The domain knowledge designer (DKD) can enter the knowledge into one or more node trees (domains). A set of domains is called a project. To start a domain, the DKD specifies the name and description, then the DKD can start building the domain by dragging the node icons from the 'Palette panel' into the domain view window, like the 'Node browser' and 'Node window' (i.e. one level view). The creation of links and groups is done analogously. After dragging an icon into a domain view window, it has to be specified by name, description, type and reference.

In the Node window the DKD can easily change the view by just double clicking on the node's box (one level lower in the node tree) or on the background (one level higher in the node tree). The 'Help panel' is always visible and helps the DKD using Organon; e.g. help on which and how actions to perform. The 'Inspector panel' provides detailed information about the active domain unit. For example, when selecting a node it presents its description, type, creation date, version number, creator and sources.

2.1.4 Future developments

As stated before the current version is far from complete. It needs further extensions and enhancements to become a useful tool. Also, the prototype is very limited, nevertheless it shows the concept. Possible future research developments for Organon are: build an operational version of Organon; evaluate the validity of the Organon concept; extend the current concept with additional ways to present the knowledge; or embed Organon into a larger environment. One larger environment could be a full instructional design environment, another could be an authoring environment for intelligent tutoring systems. At the moment we are investigating the usefulness of Organon for the intelligent tutoring system FELIX (Kuiper and van der Arend, 1991).

2.2 The NeXT computer

The NeXT computer system, introduced in 1988, has three different system compositions:

- NeXTstation
- NeXTstation Color
- NeXTcube

The NeXTstation has the technical specification: 8 Mb RAM, 3.5 inch - 2.88 Mb diskdrive, 105 Mb harddisk, 17 inch MegaPixel-display (monochroom, 1120x832), Ethernet network interface, keyboard and mouse. The NeXTstation Color has: 12 Mb RAM (expandable to 32 Mb), 3.5 inch - 2.88 Mb diskdrive, 105 Mb harddisk, 16 inch MegaPixel-display (colour, 1120x832), Ethernet network interface, Sound box, keyboard and mouse. The NeXTcube system is the same as the NeXTstation Color, except: 3 NeXTbus slots and RAM expandable to 96 Mb.

The NeXTstation has a 25 MHz 68040 Motorola processor and the interface, NeXTstep, runs on Mach, a Unix-like operating system. This is why it has the powerful networking and multi-tasking capabilites.

The NeXT systems use the Display PostScript imaging system for both the display and the printer. So what you see on the screen is unequivocally what you get on paper. Further, the NeXT has built-in CD-quality sound, allowing sound to be integrated into applications.

The NeXT has a 105 Mb harddisk with NeXTstep 2.0, the graphical interface including object-oriented Application Kit, the Interface Builder and the NeXTmail E-mailer. The harddisk can be expanded to 330 Mb, 660 Mb or 1.4 Gb, in case the 80 Mb NeXTstep 2 leaves not enough

diskspace. Optional is the 256 Mb optical disk (standard available on older NeXT systems) and the 540 Mb CD-ROM.
Many state-of-the-art applications make the NeXT an easy-to-use and user-friendly environment. Example applications are: the multimedia mail system (combining text, graphics and voice), the WriteNow wordprocessor, online dictionaries, documentation and manuals in Digital Library, FrameMaker, WordPerfect and Lotus Improve (Lotus 1-2-3 for NeXT).
If you buy a software-based PC emulator, the NeXT system can run any MS-DOS application. NeXT and DOS software can run side by side and share files as easily as copying them. They are now working on similar emulators for Macintosh environments.
Finally, the prices of the NeXT computer systems are competitive to the comparable configurations of Macintosh and the Sun SPARCstation. The Macintosh IIfx is around 1.5 times, the Sun SPARCstation 2 is around 2 times the NeXT list price.

2.3 Results

I started my work with an investigation of the projects in the Automated Instructional Development Research Team of Ray Perez. This is how I got acquainted with the Organon system. At that time, version 1 on the Macintosh was almost finished and the project team was in the brain storm session phase. After these sessions I started with Larry Brooks on the analysis of Organon version 2. The results of the analysis were documented in a report (see appendix F). When the analysis was completed Larry and I started the design phase. As a part of the Organon version 2 design, I developed a prototype of Organon on the NeXT computer. The NeXT prototype is described in the report 'The Organon prototype' (See appendix G). During my work at ARI I presented Organon, the NeXT computer and my activities to the AISTA team several times.

At the end of March, Dr. Seidel organised the 4th DoD Training Technology Technical Group (T2TG) meeting. I was invited to join the meeting of approximately 80 U.S. Defense training specialists. A brief report of the meeting is given in appendix H.
At the ARI I got the opportunity to browse through many reports (e.g. KA/IAA final report, AKAT report) and to see various systems related to training and training development (e.g. ID Expert, Authorware, Authology, IDV Technology). The characteristics and impressions of ID Expert version 3.0 are documented in appendix I.
One day in June, John Anderson of the Carnegie Mellon University presented his research on the intelligent LISP Tutor at the ARI. His presentation, titled 'Acquisition of skill in multiple

programming languages', introduced their general tutoring architecture and gave their experiences about the transfer of learning across the programming languages LISP, Prolog and Pascal.

At the end of my stay at the ARI Larry Brooks, Chris Jardine (student GMU) and I worked on the structure of a paper about the problems and possibilities of multiple-perspective knowledge acquisition for intelligent tutoring systems. We are now still working on the draft version of the paper.

3 CONCLUSIONS

The study year was very useful and instructive. At the ERAU I got the opportunity to look in detail at another intelligent training system, and its project activities, along with the FELIX system. Some of the information, knowledge and skills I already used in an intelligent tutoring system consultancy project at TNO/FEL.

At the ARI I was conducted into the field of instructional design. They presented instructional design projects, methods, tools and systems. This is very appropriate because future research projects at TNO/FEL will be in the field of instructional design. The Organon concept could also be used in the future authoring environment of the FELIX system.

REFERENCES

Arend, J. G. M. van der, and Brooks, L. W. (in press). Organon: A tool for graphical knowledge organization. Version 2: Object-Oriented Analysis II (Messages and User Perspective). ARI Working paper, U.S. Army Research Institute, Alexandria, VA, USA.

Bloedorn, G.W., W.H. Crooks, M.D. Merill, H.J. Saal, L.L. Meliza and O.I. Kahn (1985). Concept Study of the Computer-Aided ARTEP Production System (CAPS). ARI Research Report 1403, U.S. Army Research Institute, Alexandria, VA, USA.

Brooks, L. W. (in press). Organon: A tool for graphical knowledge organization. Version 1: Introduction and description. ARI Working paper, U.S. Army Research Institute, Alexandria, VA, USA.

Brooks, L. W., and Jardine, C. R. (in press). Organon: A tool for graphical knowledge organization. Version 1: Source Code and Commentary. ARI Working paper, U.S. Army Research Institute, Alexandria, VA, USA.

Coad, P., and Yourdon, E. (1991). Object-oriented analysis. Englewood Cliffs, New Jersey: Yourdon Press.

Coad, P., and Yourdon, E. (1991b). Object-oriented design. Englewood Cliffs, New Jersey: Yourdon Press.

Cox, B. (1986). Object oriented programming. Reading, Mass.: Addison-Wesley.

Gesell L.E., Air Traffic Control: An Invitation to a Career. Coast Air Publications, Chandler, Arizona (1987).

Goldberg, A., and Robson, D. (1983). SmallTalk-80: The language and its implementation. Reading, Mass.: Addison-Wesley.

Jones, Mark K., M. David Merrill and Zhongmin Li, April 1991. Implementation of an expert system for instructional design. Final report of the Strategy Analysis Component of ID Expert version 3.0. Project report with Human Technology, Inc. and The U.S. Office of Personnel Management.

Keene, S. (1988). Object-oriented programming in common lisp. Reading, Mass.: Addison-Wesley.

Kornecki A. and Edward J. Malo, Acquisition of Air Traffic Control Expertise. In: Proceedings of 34th ATCA Convention, Washington D.C., (1989).

Kornecki A. (1989) Building of an Air Traffic Control Expert System. In: Proceedings of IASTED Seminar on Expert Systems, Zurich, Switzerland.

Kornecki A. (1988) Simulation and Artificial Intelligence as Tools in Aviation Education. In: Proceedings of Eastern Simulation Conference, Orlando.

Kornecki A. (1989) Simulation Based Training - Knowledge Acquisition Problems. In: Proceedings of European Simulation Conference, Rome, Italy.

Kuiper, H., and Arend, J.G.M. van der (1991). FELIX: An intelligent tutoring system. Paper submitted to the Second International Conference on Intelligent Tutoring Systems, June 1992, Montreal, Canada.

Larkin, Don, et al. (1989). The NeXT System Reference Manual. Release 1.0 On-line Edition, NeXT, Inc., Redwood City, CA, USA.

Merrill, M. David, 1987. The new Component Display Theory: instructional design for courseware authoring. In: Instructional Science, 1987, 16, 19-34.

Merrill, M. David, 1987b. An expert system for instructional design. In: IEEE Expert, Summer 1987, 25-37.

Merrill, M. David, 1988. Applying Component Display Theory to the design of courseware. In: D.H. Jonassen (ed.), Instructional Design for Microcomputer Courseware. Lawrence Erlbaum, Hillsdale, New Jersey, USA.

Merrill, M. David, and Zhongmin Li, 1988. Implementation of an Expert System for Instructional Design (phase 2). Army Research Institute technical report. U.S. Army Research Institute for Behavioral and Social Sciences, Alexandria, Virginia, USA.

Merrill, M. David, and Zhongmin Li, 1989. Implementation of an Expert System for Instructional Design (phase 3). Army Research Institute technical report. U.S. Army Research Institute for Behavioral and Social Sciences, Alexandria, Virginia, USA.

Merrill, M. David and Zhongmin Li, 1989b. An instructional design expert system. In: Journal of Computer-Based Instruction, Summer 1989, Vol. 16, No. 3, 95-101.

Merrill, M. David, Zhongmin Li and Mark K. Jones, January 1990. Limitations of first generation instructional design. In: Educational Technology, January 1990, Vol. 30, No. 1.

Merrill, M. David, Zhongmin Li and Mark K. Jones, February 1990. Second generation instructional design (ID2). In: Educational Technology, February 1990, Vol. 30, No. 2, 7-14.

Merrill, M. David, Zhongmin Li and Mark K. Jones, March 1990. The second generation instructional design research program. In: Educational Technology, March 1990, Vol. 30, No. 3, 26-31.

Nolan, M. S. (1990) Fundamentals of Air Traffic Control. Wadsworth Publishing Company, Belmont, California.

Ransom, A. and Kornecki, A., et al. (1988) Simulation Based Expert for ATC Training. In: Proceedings of European Simulation Conference, Nice, France.

Spencer, D.A. (1989) Applying Artificial Intelligence Techniques to ATC Automation. In: The Lincoln Journal, Volume 2, Number 3.

Stroustrup, B. (1986). The C++ programming language. Reading, Mass.: Addison-Wesley.

Xerox. (1985). NoteCards. Release 1.2i. Reference Manual. Xerox Palo Alto Research Center, California, USA.

Ir. J.G.M. van der Arend (author)

Ir. H. Kuiper (project leader)

Ir. M.J. le Mahieu (group leader)

Appendix A:

THE ISTS

The Intelligent Simulation Training System

Embry-Riddle Aeronautical University

Daytona Beach, Florida

Jos van der Arend

December 14, 1990

CONTENTS

1 INTRODUCTION

During my participation in the ISTS project activities at the Embry-Riddle Aeronautical University (ERAU), I made a few notes. These notes are the structure of this report.
I made these notes to be able to record my impressions and to discuss these impressions with the other members of the project team.

This report concerns the main area of my activities: The Intelligent Simulation Training System (ISTS) project. The project, a cooperation of the University of Central Florida (UCF), General Electric Company (GE) and the ERAU, studies the usabilities and applicability of AI-techniques in an air traffic control (ATC) training system.

2 EMBRY-RIDDLE AERONAUTICAL UNIVERSITY

The Embry-Riddle Aeronautical University is a university where all the studies and lessons are related to the aviation. The university has thousands of students in different studies and activities. An important section of the ERAU is the Aviation Computer Science Department. In this department all the activities related to the computer science are gathered. All the student obtain their initial, basic and advanced computer knowledge here. Other activities of the Aviation Computer Science (ACS) department are research activities in several areas. One such a research activity is the participation in the ISTS project, the project I am taking part of.

In the AI lab houses the following equipment:

Symbolics 3630 (2 monitors): LISP oriented system for development of the ISTS prototype and text editing;

Apple Laser printer;

Gateway 2000: IBM-compatible (with 8M RAM) for VP Expert (ES shell), Clips (NASA developed ES language) and Tracon (ATC game/simulation);

IBM PS/2: for text editing (Word Perfect);

Sun SPARCstation 1+: workstation for the new ISTS prototype version to be developed and for MODSIM, XWindows, Clips applications;

3 AIR TRAFFIC CONTROLLER

Air traffic control (ATC) is a service to provide a safe and orderly flow of air traffic. The Federal Aviation Administrat.. (FAA), charged with operating the civilian air traffic control systems in the US, divided the airspace into 24 areas and assigned aircraft separation responsibility within these areas to 24 air route traffic control centres (ARTCC). Further, the FAA utilizes complex equipment located at FAA headquarters in Washington and about 400 fully equipped airport towers.

The FAA system hardware includes numerous computers along with communication, navigation, radar, ground and airborne equipment. In this environment with some 15,000 airport facilities, 400,000 miles of airways, 850,000 pilots, weather stations, military, naval and space operations the air traffic controller, often called controller, forms an important factor.

The air traffic controller, operating under the FAA, has a complex task interacting with the ATC *System* (ATCS), with pilots and with other controllers. The normal education and training time of at least 3 years before reaching its full performance level also shows the complexity of its task.

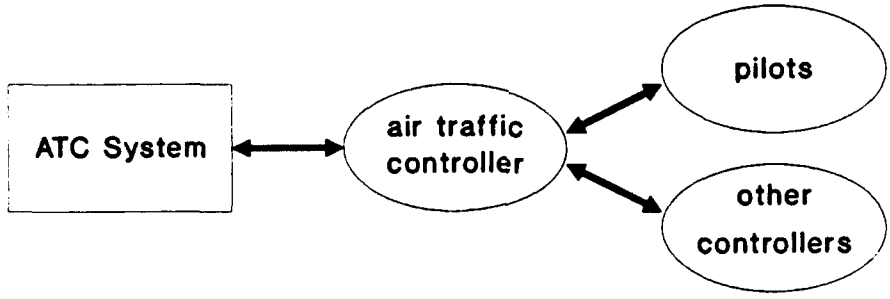

Figure A-1 Air traffic controller between system and persons

The increase of air traffic, the 1978 deregulation of airline industry and the enormous ATC personnel reduction following the strike of 1981 still place significant pressure on the system and the controllers.

To increase the controller's productivity a long-term plan (20 years) was introduced in 1981: the National Airspace System Plan. This plan must lead to the Automated En Route Air Traffic Controller (AERA). This plan contains the following stages:

1 Initial projects about:
   - ATC facilities improvement
   - Traffic Management System
   - Conflict prediction
   - Terminal improvement
   - Flight Service Station modernization
   - Navigation and Communication System improvements
   - Radar improvements
2 Advanced Automation System (AAS) project in 3 phases:
   1 automate ATCS
   2 consolidate ATC facilities
   3 improve the aviation navigation system
3 AERA project (in 3 phases)

This national airspace program is going on at least until the year 2000.

3.1 Air Traffic Control System

The basic role of the Air Traffic Control System (ATCS) in the US is to prevent collisions between aircraft whose pilots choose to participate in the ATCS. A secondary objective is to organize and expedite the safe and orderly flow of air traffic, utilizing the airspace as efficiently as possible.

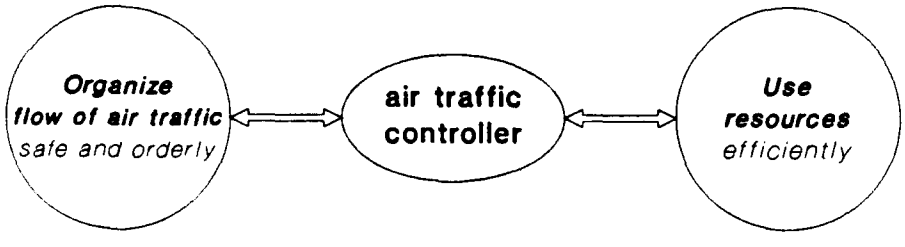

Figure A-2 Air traffic controller between his tasks

All the airspace above the US is initially classified by the FAA into two general categories: controlled and uncontrolled airspace. In uncontrolled airspace, mostly located away from airports below 1200 feet, ATC services will not be provided to any aircraft.
Pilots in the controlled airspace who wish to actively participate in the ATCS must comply with the following requirements:

- file a flight plan with the FAA
- provide their own separation from aircraft flying under visual conditions
- comply with any instruction issued by ATC unless emergency case or dangerous situation

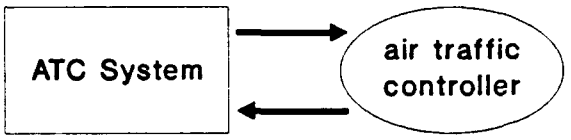

Figure A-3 Air traffic controller interfacing with the ATCS

The input of the controller into the ATC System includes:

requests from other controllers and pilots
radar display
replies from other controllers and pilots
flight strips

The output of the controller includes:

requests to other controllers and pilots
changes to radar display
replies to other controllers and pilots
changes to flight strips
keyboard and tracking ball

3.2 Air Traffic Control

According to the ATC Handbook (FAA Handbook 7110.65), air traffic controllers are required to give first priority to the safety and separation of aircraft participating in the ATCS.

The functions of the controller are:

1 Coordinating with other sectors

The basic coordination function is the hand-off of control responsibility from sector to sector as the aircraft flies along its route. A controller must not allow an aircraft to leave his sector until the controller in the next section accepts responsibility for the vehicle.

2 Navigating aircraft

Controllers are not responsible for navigation of the aircraft, unless they elect to vector one off its route onto a specified heading. In such a situation, controllers are responsible for the detailed navigation of the aircraft until it is put back onto its original route or onto an alternative route.

Examples of the phraseology used for issuing vectors are:

- Turn left heading [heading]
- Turn [number of degrees] right
- Fly heading [heading]
- Depart [fix] heading [heading]

3 Issuing altitude clearances

Before pilots can change from one assigned altitude to another, they must wait for altitude clearances from controllers. Alternatively, a controller might issue what amounts to an altitude profile in the form of a clearance with altitude restrictions.

It is important to note that although the controller specifies the time at which the aircraft may change its altitude and the altitudes that are allowable, the controller does not control the rate of climb or descent.

4 Maintaining aircraft separation

Safe distances between aircraft can be maintained either in a horizontal or vertical fashion.

Horizontal separation standards may be specified in terms of distances of times. Horizontal separation between aircraft is usually 3 to 6 nautical miles, depending on the size of the aircraft and the distance from the radar antenna.

Vertical separation are always in terms of altitudes. Aircraft at or below FL290 (is Flight Level 29,000 feet) must be separated by a minimum of 1000 feet. Aircraft operating above FL 290 must be separated by a minimum of 2000 feet. The exception occurs when two aircraft are either climbing or descending.

The separation of departing aircraft is more complex and depends on the speed, direction and destination of the aircraft.

The controller's actions can be divided into the following groups of activities:

1 Initial Clearance (before take off)

2 Departure

3 Conflict Separation

4 Hand-offs

5 Arrival

and

6 Pilot Requests and Emergencies

7 Weather and Traffic Advisories

On duty the controller meets two groups of actions. The first group is associated with the preprocessing phase, in which the inputs are acquired and preprocessed in an intelligent way, so that the controller can use the well identified information. ATC actions in this part are: acceptance of inputs, identification of the relevant inputs and recognition of a need for action.
The second group of actions is associated with the decision-making phase. In this phase the knowledge and experience of the controller are very important. ATC actions in this part are: classification of the situation, generation of an optimal course of action and output.

In ATC many factors might influence a particular decision and there are usually a lot of actions or sequences of actions that could be taken to resolve the problem. The factors vary from sector to sector and from situation to situation.
For example, when two aircraft are flying on courses that cross each other, then it seems easy to make a decision how to prevent the potential separation violation:

1 Turn left or right
Temporarily take one of the aircraft from it original flight plan by vectoring. After a specific time period the aircraft has to be placed back on its flight plan.

2 Climb or descend
let the aircraft fly at a higher/lower level to prevent a potential conflict

3 Slow down or speed up
let the aircraft reduce/increase speed to prevent a potential conflict.

4 Delay pattern
Let the aircraft make left and right curves to delay arrival time at the point of conflict

5 Holding pattern
Holding patterns are used when insufficient airspace exists for an aircraft to continue to its destination. Within a holding pattern, an aircraft is restricted to a fairly small area.

However the following things might invalidate this decision:

- there are other aircraft in the vicinity
- severe weather conditions

- a manoeuvre forces an aircraft to cross or come close to a sector boundary
- the aircraft is close to its maximum flying altitude
- the manoeuvre results in undue delays
- requiring a jet to fly long distances at low altitude wastes fuel
- it is inefficient to force a climbing aircraft to descend or a descending aircraft to climb
- if the descent of an arriving aircraft is delayed, it might not have enough time to reach the appropriate altitude.

When communicating with pilots or other controllers, a controller should always use the standard phraseology and procedures. The message must have the following format:

1 identification of the aircraft or controller being contacted
2 identification of the calling controller
3 the contents
4 termination

Certain letters and numbers may sound similar to each other when spoken over radio or telephone equipment. To alleviate this problem, a standard pronunciation of letters and numbers has adopted by the FAA. This concerns numbers (enunciated separately), altitudes, time, wind direction and velocity, runway numbers, radio frequencies, speeds, ATC facilities and route and navigation aids.

3.3 Education and Training

Before becoming employed by the FAA, prospective controllers must successfully pass a written aptitude test, a medical examination, a security investigation, and a general employment interview. The written aptitude test has three subsections:

1 assesses ATC aptitudes
2 assesses ability to perceive spatial relationship
3 assesses knowledge relating to ATC

Upon successful completion of the above steps, the applicant may be hired as a conditional employee of the FAA. Every newly hired FAA controller is required to successfully complete the controller FAA Academy Screening and Training Program. This program is designed to evaluate both the academic and practical skills of the controller. At the conclusion of this program, the Controller Skills Test is given.

Having completed the academy training program, the developmental controller is sent to an air traffic control facility for the Field Training Program. Depending on the complexity of the facility, it may take one to four years to become fully certified as an ATCer. At a control tower, the training sequence is normally flight data, clearance delivery, ground control, local control and radar control positions. Once they have completed the radar training facility and are certified on every position at the facility, they must complete a facility rating exam. After passing this exam, they are considered full performance controllers.

There are a lot of training systems/games on the market to train air traffic controllers. For example there is:

ATC 21000	from UND Aerospace
ATCoach	from UFA, Inc.
ATS 32 Simulator	from ATS (Canada)
CATS+BART+AFTN+CBT	from SCAA (Sweden) and ATS (Canada)
MAST+BAST	from Aeronautical Consulting and ATC.S (Germany)
NATSIM	from AIT Corporation
STAR	from SSI
TRACON/Pro	from Wesson International

4 INTELLIGENT SIMULATION TRAINING SYSTEM

The concept of the Intelligent Simulation Training System was originated at the University of Central Florida (UCF) in Orlando. The concept (Biegel, 1986) deals about integration of Artificial Intelligence methods and simulation. The project to proof the concept is a cooperative effort with UCF, the Control and Simulation Division of General Electric (CSSD/GE) and the Air Science and Simulation Laboratory of Embry-Riddle Aeronautical University (ASSL/ERAU) both in Daytona Beach. The air traffic control has been chosen as a test vehicle to show the applicability of the system. For implementation of the ISTS concept, the cooperating organizations were charged with the development of the elements:

- the Intelligent Tutoring Module (ITM), by UCF
- the Simulation of the ATC radar terminal (SIM), by GE
- the ATC Expert System knowledge base (ATEC), by ERAU

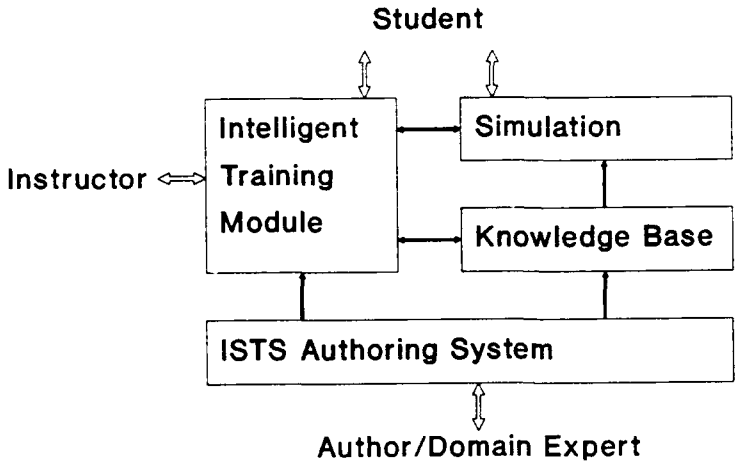

Figure A-4 The elements of the ISTS

These elements are being developed in LISP on a Symbolics 3765 LISP machine with a colour graphics system.

The project members are:

J. Biegel, M. Draman, S. Letter and students (all UCF)

K. Donovan and R. Phinney (both GE)

A. Kornecki, M. Towhidnejad, V. Galotti,

D. Hanzlik, R. Dukeshier, D. Jareckas and Jim Henderson (all ERAU)

At the moment the different partners have different objectives for the ISTS. UCF uses the training system as a system to let students get acquainted to AI and to do their tasks on. As second objective UCF tries to develop the concepts as proposed in 1986.
General Electric is interested in the final intelligent training system. They will try to sell it to the industry and maybe have some more industrial spin-offs.
The ERAU is interested in the final product as a new training medium where student of the university can train their skills. Further for the ERAU this system is a good opportunity to get familiar with AI-techniques and methods and a perfect system for graduate students.

The ISTS project is now in the stage of producing a new prototype which contains the latest version of each element from the different project partners. This prototype is planned to work in the beginning of November on the Symbolics machine. After the completion of this prototype the ISTS system is transferred to the Sun SPARCstation. The code of the programming language

LISP is converted to C++-code and the Expert System shell ART (Automated Reasoning Tool from Inference Corporation) is replaced by the ES shell Clips (from Artificial Intelligence Section of Johnson Space Center).

Further plans with the ISTS for the future are:

- Develop the prototype to a full intelligent ATC training system
- Implement a new domain in the ISTS. UCF has plans to implement the system with training for car drivers.

The ISTS is divided into 14 modules:

Author	provide the ISTS with the necessary information about the domain,
Discourse	handle all the non-simulation communications between student and system,
Translator	parse the student's simulation input into a form the system can interpret,
Input Filter	catch the student inconsistent inputs,
Diagnoser	establish the changes caused by event list, and check critical events,
Evaluator	evaluate student performance for each action,
Student Model	serve the individual needs of the student,
Tutor	conduct a lesson in a manner that best suits the student's needs,
Control	pass control to the proper modules, and assure complete communication,
Inference Engine	activate and fire rules, and provide an interface for editing rules,
Simulation	illustrate the application domain and provide 'intelligent' objects,
Intelligent Preprocessor	transfer raw simulation data into an event list,
Domain Expert	generate a set of expert solutions based on an set of simulation events,
Domain Expert Instructor	give expert instruction on what to teach with lesson plans, guide-lines, etc.

The modules can be grouped into four categories. These are:

1 Interfaces:
   - Author
   - Discourse (both UCF)

- Domain Expert
- Domain Expert Instructor (both ERAU)

2 Input:

- Translator
- Input Filter (both UCF)
- Simulation (GE)
- Intelligent preprocessor (ERAU)

3 Control Loop:

- Diagnoser
- Control
- Inference Engine (all UCF)

4 Instruction:

- Evaluator
- Student Model
- Tutor (all UCF)

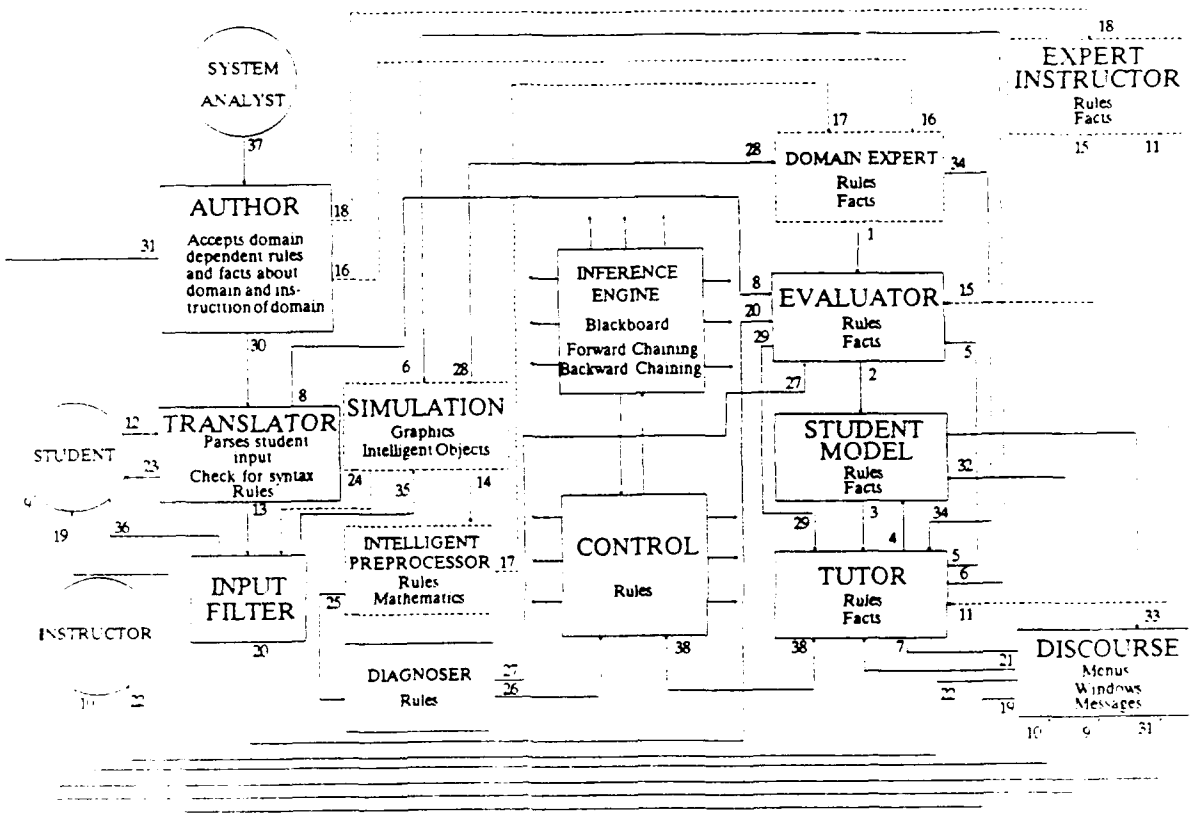

Figure A-5 The modules of the ISTS

Figure A-5 shows the relations and the connections between the ISTS modules.

5 CONCLUSIONS AND RECOMMENDATIONS

For every project there must be short-term and long-term goals; even if they are not sure or very likely to be adjusted.

Some short-term goals are evident to me: get a proper working, general accepted version of the system at the end of October; and transfer the ISTS prototype to a equivalent system on the Sun SPARCstation as soon as possible.

But, what is the main long-term goal of the project and the system? It is not clear to me where the system must lead to. Is it meant to be a research model, demonstrator, prototype or operational system? If a prototype or an operational system, what is the functionality and the deadline?

The idea of having a group of articles together is very good.
In the ASSL lies a group of articles. These articles have information concerning expert systems, AI, Simulation, ATC and any combination of these topics and this allows easy access to this kind of information by all project members. This also makes the training of new members much better and easier. However I think there are a lot of articles, certainly articles from the current year 1990, which has to be added.

For such a system with a lot of project member coming and going, it is important to have the domain exactly described.
The boundaries of the domain are clear: the domain of the en-route air traffic controller.
The VP Expert system with the ATC rules is a very good description of how the controller should react in certain situations. I think it would even be better when this could be represented on paper.
With the FELIX system we started to describe the domain on paper, before the development of the ICAI system. This was the domain of the message office clerk. Terms were defined; objects/entities and their relations were described; processes and data flows described the tasks of the message office clerk. Later, when we had an argue during meetings, we could always refer to this document.

Make rules about the documentation, program coding and about errors and modifications.
In the FELIX project:

- we had a high level and low level descriptions of each module;
- we had rules about the layout of program code (indent, 1 function name per line, maximum number of statements per routine, each routine must have a description as comment, etc.) so that it was relatively easy to read code from someone else;
- For errors we had a Error Log Form. These forms were discussed in our next project meeting.
- For modifications we had a Change Request Form. These forms were discussed in our next project meeting.

Each person of the technical team had a copy of the latest prototype in his subdirectory and was responsible for his part of the code. No one was allowed to edit someone else's code without permission.
Although the situation is different from ours (we had a network of Sun workstations with a special work breakdown structure of directories) and some of the described rules are already applied, there are some of the things I mentioned which could be useful for the ISTS project.

When there are no requirements from the client, set your own on paper in advance.
At the development of our FELIX system we had the same starting point: no requirements about the domain, the interface, the functionality or the response time of the system. Our project leader started to make milestones for the development, for example:
May 1, Design of the system on paper
Aug 1, FELIX version 1 (cosmetic prototype)
Nov 1, FELIX version 2 (limited prototype)
Jan 1, FELIX version 3 (final version)
These different versions were not only used for demonstration to the client and other interested persons, but also as a discussing subject with experts on the field of CAI, the subject matter, education, human computer interaction.
Most of these persons wrote down their comments on the system. We, the project team, decided if suggested modifications were valid and could be done within budget and time limit.

Make a clear, fast and easy to use demonstration version of the system.
As I see it, is only the 'separation violation' task covered by the system, but there are some efforts to extend this with 'need to change altitude' and 'off course handling'. When these extensions have negative influence on the response time, I would omit them in the demonstration system. The response time makes more impression on persons than the functionality.
Try to test the system with these parts exhaustively, before starting any new subjects (landing, take off, hand off, weather influence, etc.).
Try to involve some ATC students, experienced controllers or other instructors to evaluate the system and give their comments. When you are working on a system for a period of time, you are used to it and you get used to its deficits.
In the FELIX project we had an interface expert doing 'thinking aloud' (some called it 'thinking allowed') sessions. He started a student session and said everything he thought, while we recorded

his talking. For example, because he often said 'where am I' or 'what am I doing' we had to add more general overview.

System performance is bad.

The response time of the system when entering an simulation input is so long that sometimes the student can forget what button he had pressed, so he forgets what he is waiting for.

6 GLOSSARY

air traffic control	a service to promote safe, orderly and expeditious flow of air traffic.
airway	a control area in the form of a corridor.
arrival	sequence of actions related to aircraft landing.
atc clearance	an authorization by ATC for the purpose of preventing collisions between known aircraft, for the aircraft to proceed under traffic conditions within controlled airspace.
course	the intended direction of flight in the horizontal plane.
departure	sequence of actions related to aircraft taking off.
hand-off	a procedure for handling the aircraft on the border between sectors.
horizontal separation	the distance spacing of aircraft expressed in units of time or miles to achieve safe and orderly movement in flight, during landing and take-off.
nautical mile	distance size (about 1.15 statute mile or 1.85 km).
sector	a controlled airspace area under immediate supervision of one air traffic controller in charge.
separation violation	violation of the distance spacing between aircraft.
vertical separation	the separation established by assignment of different levels of flight.

7 ABBREVIATIONS

AAS	Advanced Automation System
ACS	Aviation Computer Science
AERA	Automated En Route ATC
ARTS	Automated Radar Terminal System
ASSL	Air Science and Simulation Laboratory
ATC	air traffic control
ATEC	Air Traffic Expert Controller
ERAU	Embry-Riddle Aeronautical University
FAA	Federal Aviation Administration
GE	General Electric Company
ICAO	International Civil Aviation Organisation
ISTS	Intelligent Simulation Training System
NASP	National Airspace System Plan
UCF	University of Central Florida

8 REFERENCES

Gesell L.E., "Air Traffic Control: An Invitation to a Career". Coast Air Publications, Chandler, Arizona (1987).

Kornecki A. and Edward J. Malo, "Acquisition of Air Traffic Control Expertise". In: Proceedings of 34th ATCA Convention, Washington D.C., (1989).

Kornecki A., "Building of an Air Traffic Control Expert System". In: Proceedings of IASTED Seminar on Expert Systems, Zurich, Switzerland (1989).

Kornecki A., "Simulation and Artificial Intelligence as Tools in Aviation Education". In: Proceedings of Eastern Simulation Conference, Orlando (1988).

Kornecki A., "Simulation Based Training - Knowledge Acquisition Problems". In: Proceedings of ECS, Rome, Italy, (1989).

Nolan, Michael S., "Fundamentals of Air Traffic Control". Wadsworth Publishing Company, Belmont, California (1990).

Ransom, A., A. Kornecki a.o., "Simulation Based Expert for ATC Training". In: Proceedings of European Simulation Conference, Nice, France, (1988).

Spencer, D.A., "Applying Artificial Intelligence Techniques to ATC Automation". In: The Lincoln Journal, Volume 2, Number 3 (1989)

Appendix B:

AI ASPECTS OF THE ISTS

The Artificial Intelligence Aspects

of the

Intelligent Simulation Training System

Embry-Riddle Aeronautical University

Daytona Beach, Florida

Jos van der Arend

December 14, 1990

CONTENTS

1 INTRODUCTION

In the ISTS (Intelligent Simulation Training System) project at the Embry-Riddle Aeronautical University (ERAU) are an important part of the activities directly related to the intelligence of the system. These activities can best be described by:

1 Knowledge Acquisition (KA)
2 Knowledge Representation (KR)
3 Intelligent Behaviour (IB)

I made this report to be able to record my impressions and to discuss these impressions with the other members of the project team.
In this report I will also describe these activities in the FELIX project and make a few comments about or recommendations for the ISTS project.

2 INTELLIGENT SIMULATION TRAINING SYSTEM

The ISTS is designed to be a generic intelligent training system for air traffic controllers. The ISTS certainly possess a number of AI aspects:

- It can detect conflicts of the separation violation in different situations by producing an event list with events which have to be eliminated. These different situations are created by the scenario and caused by the student himself by the simulation input.
- The system has a specific part (Domain Expert Module) that is able to generate solutions to unwanted potential events.
- The system is able to generate lesson plans, guide-lines and scenarios to teach the student (Domain Expert Instructor).
- The knowledge in the system (domain or domain instructor knowledge) is represented by facts and rules. New facts and rules can easily be added.
- To generate solutions with priorities the system uses an expert system shell (ES shell). This ES shell is the Automated Reasoning Tool (ART).
- The system is written in the well-known AI-language LISP under the modern operating system UNIX.

2.1 Knowledge Acquisition

To be able to talk on the level of controller and to elicit the air traffic control (ATC) knowledge, the ERAU ISTS-team (project manager, knowledge engineers and programmers) did a number of actions:

- study the ATC Procedures Manuals and relevant documentation,
- conduct some classes on ATC,
- interview experts (general, unstructured),
- visit the experts at work (Daytona Beach and Orlando Terminal ATC, and Jacksonville En-Route ATC),
- listen to audio tapes with recorded conversations between the controllers and the pilots,
- look at video tapes related to ATC problems.

Since the beginning of 1990 a subject matter expert (SME) joined the team. Because of his input concerning the domain knowledge the ERAU team was capable to build the expert knowledge in an expert system shell on PC. First this expert system (ES) was built with Turbo Prolog, later with the ES shell VP Expert. The expert system, built with VP Expert, can be used as an advisory tool in specific situations. The user has to describe the situations by answering questions about it. After having described the situation the system presents: the triggers, the considered ATC actions and the proposed ATC actions in the situation.
This tool is an excellent instrument to describe the knowledge in rules and facts (KR) and to elicit the knowledge from the expert.

2.2 Knowledge Representation

The knowledge in the ISTS is represented by facts and rules. These facts and rules are divided into 3 sections:

1 situation evaluation
2 domain knowledge
3 instructor knowledge

The facts and rules concerning situation evaluation are those in the Intelligent Preprocessor (IPP) which generate an event list from the current simulation variables. An example of such an event is: 'there is a potential separation violation between aircraft A and B'.

The facts and rules concerning domain knowledge are those in the Domain Expert Module. These are used to generate a set of solutions that the expert would give. This set is based on the event list from the IPP. An example of such a solution is: 'turn aircraft A heading 90'.
The fact and rules in the Domain Expert Instructor Module concern the tutoring knowledge on the domain. It gives information on how to teach the domain. The results are: teaching methods, lesson plans, guide-lines and scenarios.

2.3 Intelligent Behaviour

The system behaves like an intelligent training system. The student can turn, climb and descend aircraft and the system detects potential conflicts in every situation.
The IPP processes the simulation output and creates a factual description of the simulated air traffic situation in so-called triggers. The current system is capable to find the following triggers:

1 Predicted Violation
2 Conflict Alert
3 Need to Climb
4 Need to Descend
5 Off Flight plan

Especially the first trigger (Predicted Violation) needs an enormous amount of processing time. It calculates potential separation violations until 10 minutes before the violation time.
The solution in the system is driven by forward chaining mechanism. The controller knowledge is represented as production rules, where the situation facts constitute premises, and assertion of an action fact constitute conclusion. Searching the knowledge base and firing the rules is accomplished by the inference engine.
The system can explain and justify how the solution has been reached. It gives the student advise about possible solutions and sticks at a scenario until the student has proven the correct skills.

3 THE ICAI SYSTEM FELIX

FELIX is the Intelligent Computer Assisted Instruction (ICAI) system that the TNO Physics and Electronics Laboratory built together with system house Logica. The domain is about the message office clerk (MO-clerk). This MO-clerk works on a message office were he receives message forms. He has to check the form, register it, make it ready for sending, give the ready for sending form to the appropriate device operator (telex, radio, messenger). After having sent the message,

the MO-clerk receives a copy of the message and he has to register the sending and do some final checks.

The intelligent aspects of the system are:

- It can generate real-time solutions on the tasks the students has to learn (checking of forms, registration).
- The system is able to adapt the instruction to the level and wishes of the student. Good student don't get the learning material into much detail, the student can choose a topic to be presented.
- The development of the system was like an evolution: first an expert system, then a number of prototypes with increasing contents and functionality.
- The system is modular: Dialog Manager, Student Model, Strategy, Expert Model, I/O manager. These are the modules of the system and they are domain independent. The domain dependent part of the system is the courseware and the expert knowledge database.
- We used a modern platform: Sun workstation with SunView and Prolog and an user-friendly and easy-to-use interface (WIMP).

3.1 Knowledge Acquisition

To acquire the operational knowledge we used all kind of material concerning the subject matter, like: prescriptions, rules, procedures. We also used material concerning the current education of MO-clerks, like: the programmed instruction books and the general suggestions for the instructor how to give the instruction (syllabus).

Further we did some unstructured and structured interviews with subject matter experts and expert instructors, and we attended some classes and some practical exercises.

We used the Yourdon and NIAM (Nijssen's Information Analysis Method) methods to describe the domain on paper. We made a data dictionary, entity relation diagrams, status description diagrams and data flow diagrams.

In an early stage of the project we made a prototype of an ES in the field of the message handling. We used the AI programming language Prolog. This ES also helped us to elicit the knowledge.

3.2 Knowledge Representation

The knowledge in the system is represented by facts and rules in the Expert Knowledge base. This knowledge base is used by the domain independent Expert Model Module of the ICAI system.

All other domain dependent knowledge (sequence of topics, amount of detail, student presentation of topics, etc) is described in the courseware.

3.3 Intelligent Behaviour

Much of the intelligent Behaviour is described earlier in this section. The student gets an easy-to-use and user-friendly interface to the tutor. The student can change the sequence of instruction topics or concepts (choice-network button), he can ask the system for help on certain used words or parts of pictures (what-is button), he can ask for 1 or 2 examples of the presented concept (example button).
As a result of the real-time generation of the answer to questions from the facts and rules in the knowledge base, the system can give an explanation how the correct answer was obtained.
Beside the normal instruction cycle (presentation, question(s), tests) the student has also an opportunity to enter the so called 'expert mode'. In this expert mode the student can pose a question to the expert in the system or he can get involved in a simulated environment where he has to perform the expert task of checking messages.

4 CONCLUSIONS AND RECOMMENDATION

Every intelligent tutoring system lacks some features of the definition of a full intelligent tutoring system. The features which lack in the ISTS are:

Modern platform.
The Symbolics 3630 is not fast enough and it is not a general used computer. The system has its own LISP oriented operating system.

Easy-to-use and user-friendly interface.
Though the simulation screen looks very nice and realistic, it is not easy to handle. There is no help on the depicted buttons, screens, screen symbols, etc.
The discourse screen looks modern with the windows and buttons for the mouse, but in some way it is not easy to handle. Maybe it is the confusing screen layout, maybe it is the great amount of keyboard actions which has to be done to do what you want to do.

Advanced student adapted training.

The system can adapt to the level of the student by choosing the right lesson (and scenario) for the student. The student can adapt the training to his level by choosing the desired lesson.

For an intelligent training system this is not enough. A poor performing student will need more explanation, more examples, more counter examples, more guidance during the simulation than a good performing student. And a poor student will also need a fixed place to go to when he don't know what to do or what he is doing. Now a student can spend minutes not knowing what he is doing.

Accept a wide rang of problems.

An intelligent system should be able to handle a wide range of problems. Because the system is still being developed the range is limited in the current prototype.

When we compare the two projects ISTS and FELIX, there are a lot of differences. Some elements of the project might be used by the other system to contribute to a better system or development of the system. The elements of the FELIX project which could be of interest for the ISTS project are:

The description of the knowledge of the domain on paper.

The instruments for this approach are the data dictionary, status description diagrams and data flow diagrams of the domain.

Testing and evaluation.

The FELIX system was evaluated by many people of different origin (student users, instructors, subject matter experts, CAI experts, interface experts, courseware evaluators) in different ways (think aloud sessions, student sessions, demonstrations).

Appendix C:

MODSIM II

MODSIM II

The Language for

Object-Oriented Simulation

The Embry-Riddle Aeronautical University

Daytona Beach, Florida

Jos van der Arend

January 3, 1991

CONTENTS

1 INTRODUCTION

In the Air Science and Simulation Laboratory (ASSL) of the Embry-Riddle Aeronautical University (ERAU) is an Sun SPARCstation 1+. The MODSIM II language (version 1.4) and the SIMGRAPHICS graphical editor (version 1.1) from CACI Products Company is installed on this SPARCstation. This report gives an overview of the MODSIM language and the related graphical editor and handles about the experiences with both of them.

I made this report to be able to record my impressions and to discuss these impressions with the other members of the project team. This report can also be an important information source for the TNO/FEL about object-oriented languages in general and about MODSIM in particular.

2 MODSIM II

MODSIM is announced as the language for object-oriented programming and discrete-event simulation. It is available from CACI Products Company with locations in La Jolla (California), Washington D.C. and London (UK).

MODSIM is supported on a variety of machine architectures and operating systems. MODSIM programs are portable from one machine to another. It is available on the following computer systems: PC DOS, PC OS/2, Sun-3, Sun-4/SPARC.

The MODSIM II documentation contains three parts:

1 MODSIM II Reference Manual - The language reference. It contains information about the syntax and structure. Also covers OOP, simulation, graphics and I/O.
2 MODSIM II Tutorial - This provides an overview of the language features.
3 MODSIM II User's Manual - This contains information about: installation; release-specific information; use of mscomp and msview; compiler and debugging options.

2.1 Overview of MODSIM II

The Modular Simulation language, MODSIM II, is a general purpose, modular, block structured language which provides support for object-oriented programming (OOP) and discrete event simulation. The syntax and control mechanisms are similar to those of Modula-2, so a simple

MODSIM program which does not use the object or simulation extensions will look very much like a Modula-2, Pascal or Ada program. For instance:

```
MAIN MODULE hello;
BEGIN
 OUTPUT("Hello world");
END MODULE.
```

Programs written in MODSIM are organized around object types. Each object type has two interrelated sets of properties: fields and methods. The state of an object at any instant is described by the values in series of fields, similar to those of a record. Its behaviour, or the actions it is capable of performing, are described in its methods which are executable routines with special characteristics associated with the object type.
For instance, an object TANK is an instance of the TankObj object type. The SPEED of the TANK object is a field (f.i. TSpeed:= ASK TANK SPEED) and the STOP routine to set the SPEED to zero is a method (f.i.: ASK TANK TO STOP).

An object's field is 'read-only' from the perspective of code not included in the object, but is 'read and write' from within the object itself.

The calling entity need not have detailed knowledge of how an object will accomplish the action it is being told to do. It simply ask for a generic action to be performed and the object which receives the request message has its own tailored routine to perform that action.

New object types can be defined in terms of existing object types. When this is done, all of the fields and methods of the existing object are incorporated as a proper subset of the new object type (inheritance).

A MODSIM program consists of a main module and any number of supporting library modules. Each library module contains declarations for a set of related procedures and objects (DEFINITION MODULE) and the executable code which constitutes the procedures and methods (IMPLEMENTATION MODULE). See the example below.

```
MAIN MODULE Example;
FROM TextLib IMPORT Reverse, text;
BEGIN
 OUTPUT("Enter string:  "):
 INPUT(text);
 Reverse(text);
 OUTPUT("Reversed string: ");OUTPUT(text);
END MODULE.

{ TextLib library module }
DEFINITION MODULE TextLib;
 VAR text: STRING;
 PROCEDURE Reverse(INOUT str: STRING);
END MODULE.

IMPLEMENTATION MODULE TextLib;
 PROCEDURE Reverse(INOUT str: STRING);
 VAR
  k      : INTEGER;
  tempStr : STRING;
 BEGIN
  FOR k:=STRLEN(STR) DOWNTO 1
   tempStr:=tempStr+SUBSTR(k,k,str);
  END FOR;
  str:=tempStr;
 END PROCEDURE;
END MODULE.
```

Before you can run this program you must compile the three modules and then link them.

The simulation power of the MODSIM is in the TELL method. The ASK method is the same as the procedure call in normal programming languages. Call statements (including ASK call statements) are executed in the sequence they occur. However, the TELL method is executed asynchronously. It can elapse simulation time by the WAIT statements, which are only allowed in TELL methods. If a TELL method has a parameter list, only IN parameters are allowed.

Together with the MODSIM compiler comes the MSView (genealogy browser) and the SIMDUMP. MSView can be used for getting an overview what the fields and methods for an object are, what the parent and child objects of a certain object are, where to find the definition of an object, etc. SimDump shows you the contents of the graphical library.

2.2 SIMGRAPHICS II

SIMGRAPHICS is the graphical editor and toolkit for the good graphical capabilities of MODSIM. Using it you can easily incorporate animation, presentation graphics and graphical user interfaces into your MODSIM II programs. SIMGRAPHICS, which has to be used in combination with MODSIM, is described in the 'SIMGRAPHICS II Reference Manual'.

First there is the graphical editor. When you run this editor (command is 'simdraw'), you get three windows:

- Palette window
- Canvas window
- Tags window

The Canvas window is the draw area for the pictures. The Palette window is the functional area with buttons to create for example lines or circles or to set style or colour. The Tags window is the working area with the tags (name and graphical representation) of the created icons. Icons are the grouped graphical elements which can be loaded from or saved into a library.

With this graphical editor you can create and edit graphical elements (called icons) like: Image, Menu Bar, Dialog Window or Other Presentation Graphics. To be more specific, these elements are:

Image, containing: line, polyline, circle, filled area and text.

Menu Bar.

Dialog Window, containing: Label, Text box, Value box, Button, List box, Check box, and Radio box.

Other Presentation Graphics, like:

- Pie chart;
- 2-D plot;
- Level meter;
- Clocks;
- Text and Digital display.

An image is a group of basic primitive graphical objects to form an picture, illustration or drawing. You can change modes, styles, colours and line widths in these graphical elements. You

must group these basic objects, like lines and circles, into an icon to save it into the library. Every created icon is presented on screen by its tag (reference name and graphical representation) in the Tag window.
A Menu Bar can be used by the MODSIM program as a definition of the menu related to a window. After creation of the Menu Bar you can get an empty menu structure and you have to define the menu items. Each menu item can be activated by the program user by pressing the right button on the mouse, moving the mouse to the selected menu item and releasing the mouse button.
A Dialog Window (by the program indicated as Dialog Box) is used by the MODSIM program as a definition of the dialog to the user. This dialog is presented in a black-and-white window in front of the window it is attached to. Every Dialog Window has a tag (reference name and dialog window symbol) in the Tags window.
When you create an Other Presentation Graphics you automatically have a new icon. The tag of the icon (reference name and symbol) is added in the Tags window during creation of the icon. Once created you can change the initial values (colour, range, style, mode, label). Elements of these Other Presentation Graphics can be changed from within the program.

The second part of SIMGRAPHICS, besides the editor, is the object definitions. These object definitions can be used within the MODSIM program as a definition of the various graphical objects (icons) or part of the objects (child).
The programmer can easily use the object definitions by importing them from the corresponding module.

3 CONCLUSIONS

My general impression of the MODSIM package is good. It is object-oriented and the programmer is forced to make a modular program. Although the package has a number of disadvantages or drawbacks I still see the package as a useful and easy-to-use simulation language. The biggest disadvantage of this package are the bugs in SIMGRAPHICS and the manuals.
Very promising are future extensions of the package, like: expert systems support, network and serial communications support, support for window environments and extended graphics support.

DISADVANTAGES:

SIMGRAPHICS has bugs.

Sometimes the SIMGRAPHICS program is aborted due to a run-time error. Error messages are then 'Segmentation violation' or 'MODSIM II: bus error'.

Manuals does not look professional.

The explanation of the language and the graphical editor is insufficient and there are only a few examples in the manual. Some page numbers are missing and some references are to the wrong page number or to that missing page number.

Interface is not user-friendly.

Unlike the SIMGRAPHICS editor which looks like a modern tool, the MODSIM compiler and genealogy browser are not user-friendly. These tools are text-oriented instead of mouse-oriented.

Window capabilities are limited.

Although you can define windows of various sizes at various locations, only the biggest square that fits in the window is used. For one program, you can define only one window background colour. And this changes the background colour of the SIMGRAPHICS graphics editor too.

Menu bars are limited.

The Menu bar can contain commands only at the second level. The first level is for sorting and reaching the menu items at the second level. Menus with three or more levels are not possible.

Double mouse click is not supported.

In many applications you like to use a double mouse click to identify another selection. SIMGRAPHICS only supports the single mouse click.

Dialog windows are limited.

When a dialog window is drawn from within the program it always appears at the centre of the window it is added. This position can't be changed.

Adding colour to the boring black-and-white dialog window is also not possible.

Other limitations.

Line thickness can be varied but the differences are too big. List boxes (for the Dialog Window) must have vertical and horizontal scrollbar both.

ADVANTAGES:

Modular programming.

When you keep the modules small compilation goes fast.

Object oriented programming

Logical programs using objects and object inheritance (a new object can be defined in terms of another, already existing object)

Translatable to C.

All modules are compiled by translating them to C code; this C code is compiled. Although the translated C code is deleted an experienced user can make this C code visible.

Extendible with C procedures.

The MODSIM language can use all normal C procedures with incoming and outgoing parameters.

Easy programming language.

The MODSIM language looks like a general programming language (Pascal, Ada, Modula-2) with something extra. Therefore, it is easy to learn and to use.

Genealogy browser is very useful.

Especially when you start to use MODSIM the genealogy browser (MSVIEW) is very helpful in getting an overview in the capabilities. Experienced users use the browser to review existing objects.

Appendix D:

THE SUN PROTOTYPE OF THE ISTS

The ISTS Prototype

on the Sun SPARCstation

Embry-Riddle Aeronautical University

Daytona Beach, Florida

Jos van der Arend

January 10, 1991

CONTENTS

1 INTRODUCTION

In the ASSL lab of the Embry-Riddle Aeronautical University (ERAU) is an Sun SPARCstation 1+. On this SPARCstation is the MODSIM II language (version 1.4) and the SIMGRAPHICS graphical editor (version 1.1) from CACI Products Company installed.
Since the Intelligent Simulation Training System (ISTS) project is working on an implementation of the system on this SPARCstation, I made a prototype of the ISTS using MODSIM II. This prototype on the SPARC would shows the possibilities for interfacing with the user. The prototype is also meant to show capabilities of the SPARCstation and of the MODSIM II package. The experiences with the MODSIM package are described in another report. This report gives an overview of design and implementation of the prototype and shows possible ways for the future.

2 DESCRIPTION OF THE PROTOTYPE

The prototype is an ISTS implementation on the Sun SPARCstation in the domain of air traffic control. The prototype is developed with the object-oriented simulation package MODSIM II. The package MODSIM II contains the MODSIM II language (version 1.4) and the graphical tool kit SIMGRAPHICS II (version 1.1). It runs under Unix-like SunOS (release 4.1) in combination with the window environment SunView.

2.1 Functional description

With the ATC prototype the user is able to practice the skills of the en-route air traffic controller. The user (or student) gets on the screen:

1 plan view display,
2 three flight strip bays,
3 window for the display of the incoming messages,
4 icon representing the communication to the pilot,
5 icon representing the communication to the other controllers,
6 simulation time.

These different elements are divided on screen in the following way:

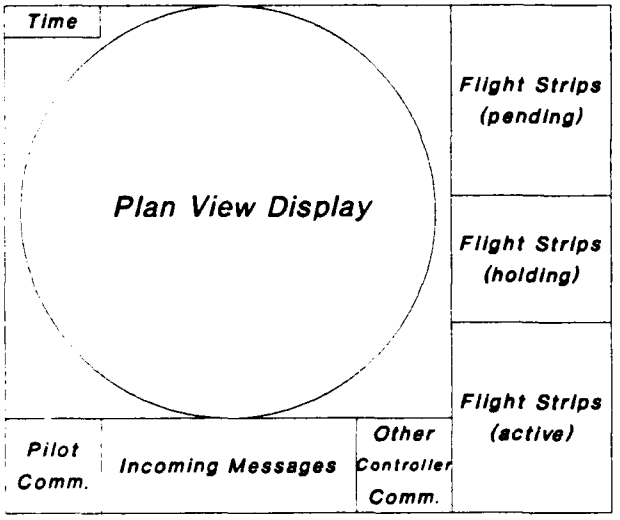

Figure 1 Screen layout

On the scope the user sees a radar image of the sector he/she has to control. In the current version is that the high altitude sector Lake City (FL 240 and above) from the Jacksonville ATC Centre in Florida. The description of the airspace around this sector (fixes, airways and airroutes) come from the Airspace file.

The aircraft are being generated from the Scenario file or randomly, according to the choice of the user.

The prototype is able:

1 to let the user manipulate the flight strips, like:
   - update the contents of the strip;
   - moving a strip from a bay to another;
   - changing the sequence of the strips in a bay;
   - deleting a strip.
2 to let the user communicate with pilots, by construct and send a message to a pilot.
3 o simulate the communication from pilots, by responding to requests from user to pilot.
4 to simulate the communication from other controllers, by responding to requests from user to controller.
5 to simulate the execution of controller actions, like: hand off procedure to and from the sector.

6 to simulate the execution of aircraft actions, like:
- climbing to a higher altitude;
- descending to a lower altitude;
- turning the aircraft to a specified heading (left or right);
- flying to a specified fix;
- hold the aircraft at the current position.

2.2 Technical description

The executable of the prototype is called 'atc' and it includes the following data files:

'atc'	the executable code;
'atc.lib'	library with graphical objects (like: images, dialog boxes and menus);
'Airspace.dat'	data file with definition of the airspace (sector data, fixes, airways and routes);
'Scenario.dat'	data file with definition of the scenario (global scenario data and aircraft descriptions)

The source code of the prototype contains the following source files:

'Matc.mod'	Main module MODSIM code;
'DIntro.mod' and 'IIntro.mod'	Intro module MODSIM code;
'DAirspace.mod' and 'IAirspace.mod'	Airspace module MODSIM code;
'DPVDisplay.mod' and 'IPVDisplay.mod'	PVDisplay module MODSIM code;
'DComDisplay.mod' and 'IComDisplay.mod'	Comdisplay module MODSIM code;
'DFSDisplay.mod' and 'IFSDisplay.mod'	FSDisplay module MODSIM code;
'DPlane.mod' and 'IPlane.mod'	Plane module MODSIM code;
'DScenario.mod' and 'IScenario.mod'	Scenario module MODSIM code;
'atc.lib'	graphical library;
'Airspace.dat' and 'Scenario.dat'	data files;

The MODSIM language code (all the files with extension '.mod') looks like the Ada, Pascal or Modula-2 language code.
All modules, except the main module 'atc', consist of a definition module source (names starting with 'D') and an implementation module (names starting with 'I').

I will give a short description of the different modules and the required data files.

Main module atc.

This is the main module; the module with the main program. This main program calls the different initialisation procedures (Introduction, InitAirspace, InitPVD, InitComDisplay, InitFSDisplay), tells the scenario to generate the aircraft (TELL Scenario TO GenerateAirplanes) and starts the simulation (StartSimulation). The simulation runs now until there is no aircraft anymore. It imports the initialisation procedures from the associated module.

Module Intro.

This module contains the introduction procedure 'Introduction' which displays the introduction screen and the handles the dialog box window to set the simulation parameters (like: Scenario generated from file or randomly). It also contains the definition and implementation of the local objects 'ACObj' and 'DBObj'.

Module Airspace.

This module contains the definition and implementation of the 'AirspaceObj' and the procedure 'InitAirspace', which reads the airspace data file ('Airspace.dat'), and puts the data in the Airspace object.

Module PVDisplay.

This module contains the definition and implementation of the procedure 'InitPVD', which displays the image of the Plan View Display in the upper left window on screen, and of the related objects like the window menu bars ('PVDMenubarObj', 'HComMenubarObj', 'TComMenubarObj') and the input data dialog box 'IDDialogBoxObj'.

Module ComDisplay.

This module contains the definition and implementation of the procedure 'InitComDisplay', which displays the three communication windows (Headset, Incoming messages and Telephone) on the lower left of the screen, and of the related objects 'HeadsetObj', 'MsgListBoxObj' and 'TelephoneObj'.

Module FSDisplay.
This module contains the definition and implementation of the procedure 'InitFSDisplay', which displays the three flight strip windows (Pending, Holding and Active) on the right of the screen, and of the related objects 'FSWindowObj', 'FlightStripObj', 'FSDialogBoxObj' and 'FSMenubarObj'.

Module Plane.
This module contains the definition and implementation of the object 'PlaneObj'.
Because this object represents the aircraft and so it is an very important one, this object has a lot of ASK and TELL methods. This makes this module the biggest of all the modules.

Module Scenario.
This module contains the definition and implementation of the object 'ScenarioObj'.
The scenario object contains the scenario data in its fields and has the TELL method 'GenerateAirplanes' to operate the generation of aircraft (from file or randomly).

Graphical library atc.
This library contains the graphical objects made with SIMGRAPHICS II and used by the prototype. The MODSIM code reads the data from this library real-time, so changes can be made to the library after compilation of the MODSIM code but the links from the code to the elements in the library are kept unchanged.

Airspace data file.
This file contains the data necessary for the initialisation of the airspace (procedure 'InitAirspace' of module 'Airspace'). It consists of the following data:

1 Fixes, with for every fix:
   - fix name;
   - fix type (VOR or not);
   - coordinates of fix position.

2 Sector data.
   This contains now only the number of the sector to be controlled. In the future the sector boundaries can be added.

3 Airways.

Description of the airways of the various routes, with for every airway:

- airway number;
- sector number of previous sector;
- sector number of next sector;
- coordinates of the 10 points (maximum) per airway:
  1) point to enter the scope (appear point);
  2) point of fix 1 before entering the sector;
  3) point of fix 2 before entering the sector;
  4) point of entering the sector;
  5) point of fix 1 inside the sector;
  6) point of fix 2 inside the sector;
  7) point of leaving the sector;
  8) point of fix 1 after leaving the sector;
  9) point of fix 2 after leaving the sector;
  10) point of leaving the scope (disappear point).

4 Routes.

Description of the airroutes related to the sector, with for every route:

- route number;
- airway number of related airway;
- entry altitude;
- exit altitude;
- flight strip with origin, route description and destination.

Each of the four parts of this data file must be preceded by one or more comment lines. Inside these parts may not occur any comment line. Comment lines are lines starting with a semicolon.

Scenario data file.

This file contains the data necessary for the generation of the scenario. It consists of the following data:

1 Global scenario data, like:

- scenario number;
- random seed for the random generator;
- speed rate of the moving aircraft.

2 Aircraft specifications.

The scenario has to contain one or more aircraft which are moved over the scope (PVD) according to the specifications. The aircraft specification contains:

- route number of airroute;
- optional aircraft generation types, like:
  1) air speed of the aircraft;
  2) aircraft type;
  3) altitude of the aircraft;
- interval time (number of seconds to wait before generating the next).

Each part (global scenario data or one aircraft specification) of this data file must be preceded by one or more comment lines. Inside these parts may not occur any comment line. Comment lines are lines starting with a semicolon.

3 LIMITATIONS OF THE PROTOTYPE

The limitations of the prototype and the possible solutions or directions for the future are:

Taking an aircraft off its flight plan is possible, but still limited and the way it is technically realised is not flexible.

Once an aircraft hes been taken off its flight plan, it has to return to the flight plan (by cleared direct) within the sector and to the fix inside the sector it was flying to or the next fix inside the sector.

In the future, data, that now have to be entered, will have to be calculated by the program.

The airspace designer has to input points like entry and exit points of the scope and the sector (in 'Airspace.dat'), but the prototype should be able to calculate these points from the route data in some method.

This method could be used in the future to calculate the changed point of leaving the sector/scope after an aircraft has been taken of its flight plan (by turning, cleared direct to fix or hold).

There is no analysis or judgement of the performance of the user (or student) in this prototype.

This function can be implemented by introducing new objects like:

- 'ExpertObj',
  with methods like 'SearchForPotentialConflict' and 'FindSolutionsToAvoidConflict';

- 'InstructorObj',
  with methods like 'GradeStudentAction', 'SetNextScenario';
- 'StudentObj',
  with fields like 'StudentName', 'CourseType', 'Scores', 'History';
  with methods like 'UpdateScores', 'DisplayScoresToUser'.

The way an aircraft (instance of object 'PlaneObj') is taken off its route is not very clear and not easily done.

Taking an aircraft off its flight plan is now done by interrupting the original tell method ('MoveAlongRoute' of object 'PlaneObj') or the off flight plan taken tell methods ('TurningLeftHeading', 'TurningRightHeading', 'ClearedDirectTo' or 'HoldAt' of object 'PlaneObj').

When the original tell method 'MoveAlongRoute' is interrupted it sets the value of the field 'InterruptStatus' of the object 'PlaneObject' from 0 to a value above 0. This value indicates the position of the aircraft according to the points of its route. Then this method 'MoveAlongRoute' waits until the value of 'InterruptStatus' is set back to 0 (by the interrupting method).

Once this is done the method completes the 'MoveAlongRoute' by moving the aircraft to the point to leave the sector and further.

The interrupting methods (like 'TurningLeftHeading') interrupt the tell method 'MoveAlongPath' whenever this method is executed and the boolean variable 'ReadyForOffPlan' (of object 'PlaneObj') is True. In this prototype these methods are executed whenever the user sends a correct message to the pilot to do so. These interrupting methods bring the aircraft to the position on the route (the last fix in the sector) from where the original tell method 'MoveAlongRoute' can take over. It sets the field 'InterruptStatus' back to 0 and 'MoveAlongRoute' takes it over again.

So, the value of the field 'InterruptStatus' of the object 'PlaneObj' represents the status of the tell method 'MovingAlongRoute' when is was interrupted. If the value of 'InterruptStatus' is:

0 then there is no interrupt or no interrupt anymore;

> 0 then the method 'MoveAlongRoute' is interrupted, but when:

<110	then the aircraft is moving to the sector;
110	then the aircraft is moving to the fix inside the sector;
120	then the aircraft is moving to the second fix inside the sector;
>120	then the aircraft has passed the last fix inside the sector;

Dialog box window limitations.

The dialog box windows are always presented in the middle of the window it is attached to, it cannot be presented on a certain position in the window. This is due to MODSIM.

In the prototype the dialog box windows are presented at the desired position on screen by attaching them to a dummy window and display that dummy window behind a window, so that you can't see this dummy window. This is done for the 'Input Data' and the 'Verbal Message to Pilots' dialog box, which are presented in the lower left and right corner of the 'Plan View Display' window.

Maybe future releases of MODSIM will solve this problem.

Only the biggest square in the centre of a window can be used for images.

This disadvantage of MODSIM leaded to the three communication windows (instead of one) in the lower right corner of the screen. This also leaded to the three 'Flight Strip Bay' windows on the right of the screen (instead of one). The caused another problem because we wanted the 'Active' and 'Pending' bay to be bigger than the 'Holding' one. But if we make this 'Holding' bay smaller, then the presented flight strips become smaller (only the biggest square can be used). We solved this problem by presenting the 'Holding' partly behind the 'Pending'.

Maybe future releases of MODSIM will solve this problem.

Some of the tell methods contains Wait statement which are not required by the design but which are required by the code.

When the execution of a tell method is aborted with the error 'MODSIM II: Segmentation Violation', then this can be solved by putting more wait statements between the other statements.

The menu bar is limited.

The menu bar (the menu which pops up in a window when clicked on the right mouse button) is allowed to have commands only at the second level. When the user chooses a menu item from the first menu level (by releasing the button at this first level) then the first menu item of the second level of the chosen item is executed.

When you want a menu that executes the commands at the first level then you have to do it in MODSIM the same way as we did: repeat the item at the second level and make the label of that second level item empty (or repeat it).

Maybe future releases of MODSIM will solve this problem easier.

Further enhancements.

The scope (PVD) has no sweeper. Introduction of this object with the tell method Turn (a loop to display the turn of the sweep a certain angle and to wait for the next turn to be displayed).

No speed of the aircraft is related to the type of the aircraft of to the accelerations it is doing.

And, wind effects are only very limited implemented. It is done by varying the true air speed randomly around the ground speed.

Appendix E:

CBT IN THE ISTS

THE CBT ASPECTS OF

THE INTELLIGENT SIMULATION TRAINING SYSTEM

Embry-Riddle Aeronautical University

Daytona Beach, Florida

Jos van der Arend

January 10, 1991

CONTENTS

1 INTRODUCTION

The Intelligent Simulation Training System (ISTS) project at the Embry-Riddle Aeronautical University (ERAU) produces a training system with intelligent capabilities. The problem with the system as it is now is that beginning students do not know what they are supposed to do or they do not have the required knowledge to act as an air traffic controller. The system cannot train students who don't have the basic knowledge of ATC or lack the capability to work with the system.
The solution to this problem might be to extend the training system with an instructional part. This part will operate as a computer based instruction (CBI), also known as computer assisted instruction (CAI), to learn the student the factual, conceptual and procedural knowledge.

In this report I will give an initial design proposal for the CBT in the ISTS.
When it names some topics in this design, it does not contain the exact and exhaustive list of topics. I leave this task to the experts in the field (subject matter expert or expert instructor).

2 INTELLIGENT SIMULATION TRAINING SYSTEM AND CBT

The ISTS is designed to be a generic intelligent training system for air traffic controllers. If the system is really intelligent it should be able to train students of different levels. For this purpose the ISTS should contain a CBT part. The CBT part will be the interface of the system to the student, but behind this CBT there is an intelligent part (expert controller and expert instructor).

When the student enters the CBT lesson the following main menu appears with the options:

1 Basic ATC Lesson
2 Procedural ATC Lesson
3 Guided ATC Training
4 Coached ATC Training
5 Free ATC Training

6 Lesson Overview
7 Results Summary
8 Stop

The first five options (1-5) represent the sequence to go through the lessons in a way of increasing difficulty. All five lessons start with an explanation of the contents of that lesson.

2.1 Basic ATC Lesson

The first option (Basic ATC Lesson) is an introduction to the profession of air traffic controller and some topics are taught like (from ATC Handbook, Chapter 2):

- Flight Plan,
- Flight Progress Strip,
- Data Block,
- System Operations,
- Route/Navaid/Command Description,
- Abbreviations,
- Phraseology

The average student should do this lesson in about 1 hour.

2.2 Procedural ATC Lesson

The second option (Procedural ATC Lesson) is the lesson about the different procedures the controller has to know like (from ATC Handbook, Chapter 5):

- Hand-off,
- Altitude Clearance,
- Vectoring,
- Direct Routing,
- Conflict Resolution (by altitude clearance, by vectoring, by direct routing, by speed control, by holding),
- Conflict Alert,
- Response to requests (from Pilots, from other Controllers),
- Standard Operating Procedures,
- Departures/Arrivals

In this lesson the procedures are explained (when they occur, why they occur, what the controller must do and what he can do, etc.). In this lesson are no simulations used, but the appropriate screen pictures and displays may be used to explain the procedures.

The average student should do this lesson in about 1 hour.

2.3 Guided ATC Training

The third option (Guided ATC Training) is a part where the student can act as a real controller. The student get a representation of the ATC system (radar display, flight strips, communication devices, etc) and the aircraft come into his sector according to a scenario. The scenario is fixed and set up by the system from the scenario file.

Each scenario deals with a specific procedure from the Procedural ATC Lesson or deals with this procedure in combination with procedures already taught. A scenario must describe the airspace for only a limited time (5-15 minutes) and for the same procedure can scenarios of different levels be used. The ways to differentiate within the same procedure is the number of aircraft and the amount of time to apply the procedure or the number of other procedures to be applied.

Before the scenario is started, the student gets a presentation of the objectives for this scenario and a short description what he is supposed to do.

During the scenario the student can always stop the simulation and ask for a brief or detailed description of the knowledge or procedure to be applied.

The average student should do this lesson in about 1.5 hours.

2.4 Coached ATC Training

The fourth option (Coached ATC Training) is a part where the student can act as a real controller in the same way as with the Guided Training.

The different scenarios for this option are fixed, but the system chooses the most appropriate scenario. Each scenario deals with a specific procedure from the Procedural ATC Lesson or deals with this procedure in combination with procedures already taught. A scenario must describe the airspace for only a limited time (10-20 minutes) and for the same procedure scenarios of different levels can be used.

Before the scenario is started, the student has no idea what he is supposed to do (which procedure he should apply).

During the scenario the student can always stop the simulation and ask for a brief description of the procedure to be applied.

The average student should do this lesson in about 1.5 hours.

2.5 Free ATC Training

The fifth option (Free ATC Training) is a part where the student can act as a real controller in the same way as with the Guided Training. Only now the airplanes are generated randomly. We assume that when the student acts as a controller in this way for a relatively long period (1 hour) then he should apply all the different knowledge and procedures.
During the simulation the level to be able to handle all the aircraft is updated in relation to the performances of the student. This level can be adjusted by changing the number of aircraft and so the complexity of the different actions. The system should also be able to adjust the scenario to the repair of the wrong or inadequate knowledge of the student. For instance, when a student performs bad at Altitude Clearances, this part should be trained.
During the scenario the student can never stop the simulation, but he can ask for a brief description of certain to be applied procedures.
The average student should do this lesson in about 1.5 hours.

2.6 Lesson Overview

The 'Overview'-option is to give new and experienced student a facility to overview the total contents of the CBT lesson. This option should take the user about 5 minutes.

2.7 Results Summary

With the 'Results Summary' the student can see how far he is and how he performed (global and detailed score).

2.8 Stop

With the 'Stop'-option the student can stop his session. All the data needed to go on the next time is saved.

Appendix F:

ANALYSIS OF ORGANON

ORGANON: A TOOL FOR GRAPHICAL KNOWLEDGE ORGANIZATION

Version 2: Object-Oriented Analysis I
(Classes and Objects)

J. G. M. van der Arend
TNO Physics and Electronics Laboratory at The Hague,
The Netherlands

and

L. W. Brooks
U.S. Army Research Institute at Alexandria, Virginia

U.S. ARMY RESEARCH INSTITUTE
5001 Eisenhouwer Avenue, Alexandria, Virginia 22333-5600

June 1991

ORGANON: A TOOL FOR GRAPHICAL KNOWLEDGE ORGANIZATION. VERSION 2.0: OBJECT-ORIENTED ANALYSIS I (CLASSES AND OBJECTS)

CONTENTS

LIST OF FIGURES

1 INTRODUCTION

In the modern Army, personnel are increasingly faced with the task of solving complex problems that require a great deal of expertise. These problems are referred to as being knowledge rich because only experts, with a large amount of information can solve them. Additionally, many of these problems are ill defined in that they can not be specified in terms that allow for the characteristics of their solutions to be known ahead of time (e.g. VanLehn, 1989). Some examples of the kinds of problems that fall into this category of being knowledge rich and ill defined are: analyzing an task so that training material can be developed for that task; collecting and organizing human expertise so that an expert system can be built based on that expertise; analyzing intelligence information so that subtle aspects and connections among various elements can be found. In particular, the work reported in this paper is part of an effort to enhance the design and development of collective training documentation (ARTEP). Currently, the construction of ARTEPs is a paper based process; however, since creating an ARTEP is a knowledge rich, ill defined problem solving process, it is an appropriate candidate for an experimental development effort aimed at facilitating this process through automation.

This paper presents the software analysis for a system, Organon, that will be able to assist humans in organizing and analyzing complex bodies of information for the purpose of solving knowledge rich and ill defined problems such as the construction of ARTEPs. Succinctly, Organon is a software tool that lets users enter and organize information using a graphical interface built around the notion of nodes and links (Xerox, 1985). A previous, prototype version of Organon exists (Brooks, in press; Brooks & Jardine, in press). However, in the course of developing the prototype it became obvious that a second version of the software would have to be built in order to meet the requirements of the project. These requirement are:

(1) The software would have to meet some of the basic needs for expert problem solvers such as the ability to chunk information, to view knowledge from multiple viewpoints, and to work on a problem in a top-down and a bottom-up manner;

(2) The software must serve as a foundation on which domain dependent knowledge organization tools could be built;

(3) The software must allow for the relatively easy extension of its basic capabilities so that future enhancements to the system will not require the underlying software to be re-written; and

(4) The software should be written so that it is oriented toward having a group of people work on organizing the same knowledge base.

To some degree all of these requirements have been met by the analysis reported in this paper and its companion (van der Arend and Brooks, in preparation). The first requirement is met by keeping and expanding the features present in the first Organon prototype (These features are discussed briefly in the next section of this paper). Enhancements to the prototype include: the use of domains as a high level knowledge organization technique; the use of virtual nodes to allow for complex information webs to be built using a simple user interface, and additional changes that increase the utility of the software for use by a group. Requirements 2 and 3 are primarily met by using an object-oriented approach to analyzing Organon. The use of object-oriented analysis and design facilitates both the expansion and reuse of existing (once it is created for the new version). The last requirement on providing more group-oriented features serves more as a pointer to future development than it does as a criteria for the existing analysis. Nevertheless, this requirement was addressed in part by explicitly having the users of the system identify themselves and the other members of the team working on a particular project.

In order to give the reader the appropriate background for understanding the software analysis presented in this paper, in the next two sections, we will: (1) briefly review the existing Organon prototype, and (2) present an introduction to the object-oriented analysis method that we used. After that we will discuss the Organon analysis with an emphasis on the classes and objects that we have defined for the system. In a second paper (van der Arend & Brooks, in preparation) we will continue the analysis by presenting the message structure and the user's perspective of the software system.

2 THE ORGANON PROTOTYPE (VERSION 1.0)

In this section, a short overview of the Organon prototype that served as the basis for the current analysis is presented. This overview is not intended to be complete, but to give the reader a feel for what a working prototype of version 2 of the system would be like. The existing prototype for Organon version 1 is implemented in Common Lisp on the Macintosh II family of computers. Originally, the Organon project began as an attempt to create a knowledge engineering environment called KEEN (Knowledge Engineer's Electronic Notebook). As the project progressed, it became apparent that a more fundamental tool for organizing knowledge or

information for a variety of purposes should be constructed. This led to a change in emphasis during the development of the prototype and directly influenced our thinking in doing the analysis reported here.

The Organon prototype allows users to construct a graphical network of nodes and links where the nodes can represent concepts, facts, rules, etc., and the links represent named associations between the nodes.

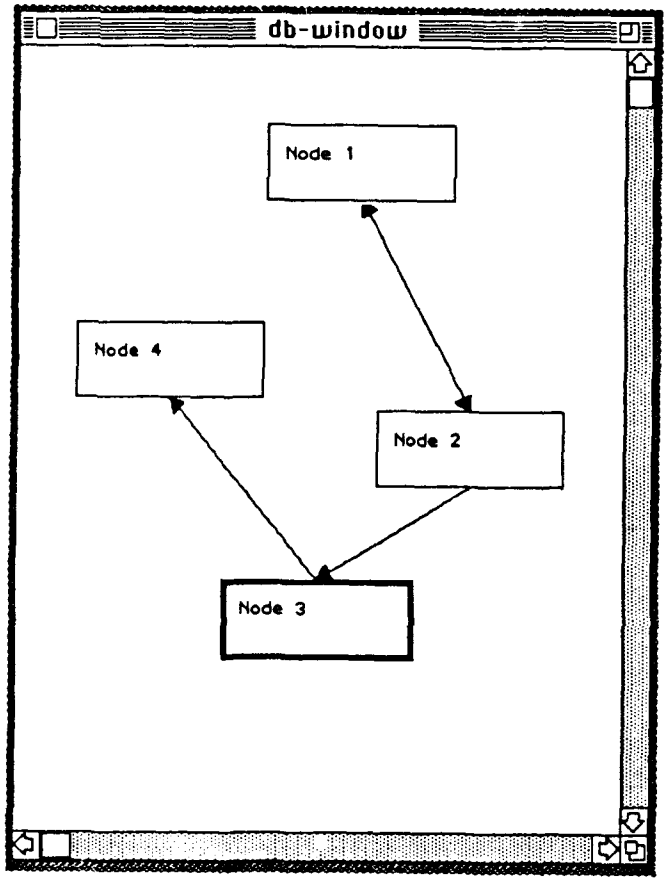

Figure F-1. Node Graphic Window

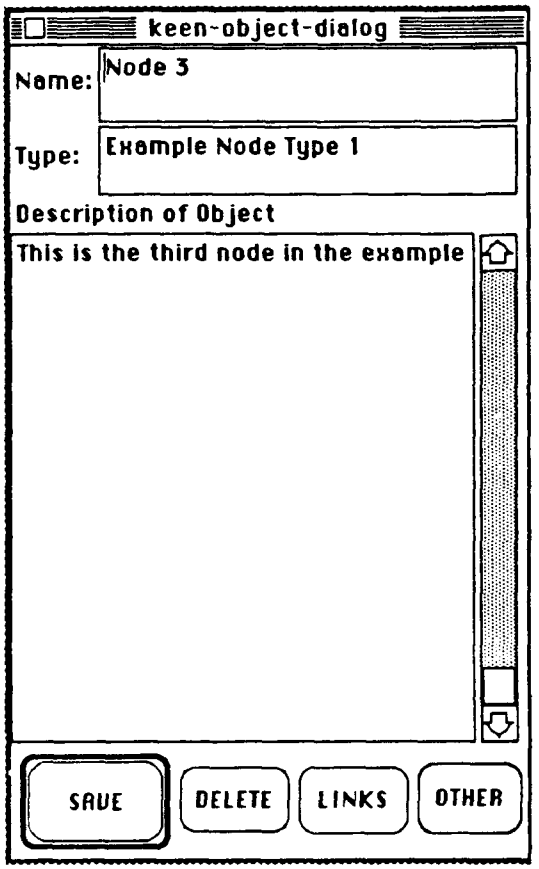

Figure F-2. Node Attribute Window

Figure F-1 is a screen image taken from an Organon session that shows a simple node network. Each node contains slots or attributes pertaining to the node's name, type, and description. These slots are user accessible and can be modified at any time by the user. Figure F-2 is a screen image of the interface used in the prototype to view and change the information contained in these slots. Additionally, there are slots for each node that are hidden from the user and are used by the system. These slots contain information concerning the placement of the node's visible representation on the screen (also called the node's box), the other nodes to which the node is linked, the parent and child nodes, etc.

To give the reader a feel for how the prototype works, a few user interactions are described in some detail. To create a new node the user double clicks in the Organon graphic window (Figure F-1) with the mouse button. A pop-up dialogue window appears immediately after the double click. This window asks the user to provide a name, type, and description for the node. The new

node is then placed on the window at a location where the mouse cursor was at the time of the double click.

The creation of a new link is similar. A new link is made when the user double clicks the mouse button while the mouse cursor is inside an existing node. Then, continuing to hold down on the mouse button after the second click, the user moves the mouse cursor to another node. A line is drawn from the first to the mouse cursor while this takes place. Next, a pop-up window appears on the screen and requesting the user to enter information concerning the type of link to be created. If the type is not known, then a request is made to the user to provide a text description of this type of link. Last, if the type of node is new to the system, then the type is added to the links menu in the top menu bar.

Finally, to move and relocate nodes on the screen, the user clicks the mouse button once while the cursor is inside a node, and then holding the mouse button down, moves that node to the desired position. When the mouse button is released, the node is re-drawn in the position occupied by the ghost box, and all connected links are re-drawn to the moved node.

The prototype allows for a greater range of action than those just described, but these actions are the ones that are central to understanding how the software to be built using the analysis presented here should work. For a more detailed description of the prototype, the reader is referred to Brooks (in press).

3 INTRODUCTION TO OBJECT-ORIENTED ANALYSIS

To understand the analysis of Organon version 2.0, familiarity with object-oriented analysis (OOA) in general, and the particular OOA technique that we used is necessary. Therefore, a brief overview of object-oriented programming and OOA is first presented, followed by an introduction to the OOA methodology used in our analysis.

3.1 Object-Oriented Programming

The general notion underlying the object-oriented approach is that software can be thought of and developed from a data perspective as opposed to a procedure perspective. That is, instead of thinking of a computer program as a collection of procedures that are called on to operate on

various pieces of data, one can think of a program as a collection pieces of data that are called on to operate on themselves. Some of the most important characteristics of the object-oriented approach are:

(1) A data module or unit is called an object, and objects have associated functions (sometimes called generic functions) that know how to perform operations on a certain object. Objects are typically though of as data structures that have slots that contain values. These values can be read or changed by an objects associated functions.

(2) Objects belong to classes. A class is an abstract data type that can be thought of as a template for a group of objects that captures the commonalties among those objects. Objects that belong to class are often called instances of that class.

(3) Classes are structured in a hierarchical manner, and characteristics of higher level classes can be inherited by lower level classes. These inherited characteristics include slots, slot values, and functions.

(4) Objects communicate by passing messages to themselves and to other objects. Messages typically ask an object to perform some action on itself via an associated function. Messages area also called methods.

(5) How an object is structured and how functions work on that object is invisible to other objects. In other words, there is no need to know the internal workings of an object in order to use it. This is called encapsulation.

For a more complete introduction to the object-oriented approach to programming in specific languages, the reader is referred to the following: (1) Lisp - Keen (1988), (2) C++ - Stroustrup (1986), (3) Objective C - Cox (1986), and SmallTalk - Goldberg & Robson (1983).

3.2 Object-Oriented Analysis

Object-oriented analysis (OOA) is a way of analyzing a problem domain taking advantage of the benefits of an object-oriented approach. In this respect OOA is similar to older software analysis

techniques that have been developed for other styles of programming such as structured programming. Some examples of these methods are:

1. Functional Decomposition.
   In this method the new system is decomposed by selecting what processing steps and sub-steps are required. The analyst then specifies the interfaces. The final specification describe the system, subsystem, functions and sub-function levels.
   Functional decomposition only indirectly maps back to the subject matter, and is therefore difficult to construct, and it is highly volatile. In OOA, functional decomposition is used for the specifying the services.

2. Data Flow.
   With this method, the analyst maps from the real world into data flows and bubbles by following the flow of data. The bubbles are hierarchical grouped, where the highest level is the context diagram representing the entire system. Additional documentation includes the specifications of the data flows (data dictionary) and the bottom-level bubbles (process specifications). Sometimes an information model (like: entity relationship diagrams) are used too.
   Problems with this method is how to pick the bubbles, the size of the data dictionary and the weak data structure emphasis.

3. Information Modeling.
   The strategy in this method is to find objects in the real world and describe them with attributes. Further add relationships, refine with supertypes/sub-types and associative objects and then normalize. Compared to the OOA, information modeling is incomplete. The following concepts are absent: services, messages, inheritance and structure.

OOA builds upon each of these techniques. From information modeling come constructs analogous to attributes, instance connections, generalization-specialization, and whole-part. From OOPL and KBS come the encapsulation of attributes and exclusive services, communication with messages, generalization-specialization, and inheritance.

The particular OOA method that we used in our analysis is the one suggested by Coad and Yourdon (1991). We choose this method because of its simplicity and language independence.

From this point on, when we use terms associated with OOA, we specifically mean the version of OOA prescribed by Coad and Yourdon. The Coad and Yourdon (C & Y) model consist of five layers (see Figure F-3):

---------- Subject layer ----------
---------- Class-&-object layer ----------
---------- Structure layer ----------
---------- Attribute layer ----------
---------- Service layer ----------

Figure F-3. The multi-layer model

The five major activities of OOA are: (1) Finding class-&-objects, (2) Identifying structures, (3) Identifying subjects, (4) Defining attributes, and (5) Defining services. It is helpful to move from one activity to the next, but it's not mandatory. The analyst may do the activities in the sequence he or she prefers. The next paragraphs describe the actions in more detail.

Finding class-&-objects. The first activity is to find the objects that belong within the problem domain and the system's responsibility. Before describing these activity, however, an expansion on the earlier definitions for class and object may be helpful. The definitions of object, class and class-&-object specific to the C & Y model are:

object, an abstraction of something in a problem domain, reflecting the capabilities of a system to keep information about it, interact with it, or both; an encapsulation of attribute values and their exclusive services (synonym: instance);

class, a description of one or more objects with a uniform set of attributes and services, including a description of how to create new objects in the class;

class-&-object, a class and objects in that class.

The class-&-objects are the framework for analysis and specification. Choose class-&-object names using standard vocabulary for the problem domain and use readable names (singular noun, or adjective and noun).

To find the class-&-objects, investigate the problem domain: observe first-hand; listen actively; check previous OOA results; check other systems; read, read, read; and prototype.

Look for structures, other systems, devices, things or events remembered, roles played, operational procedures, sites, organizational units.

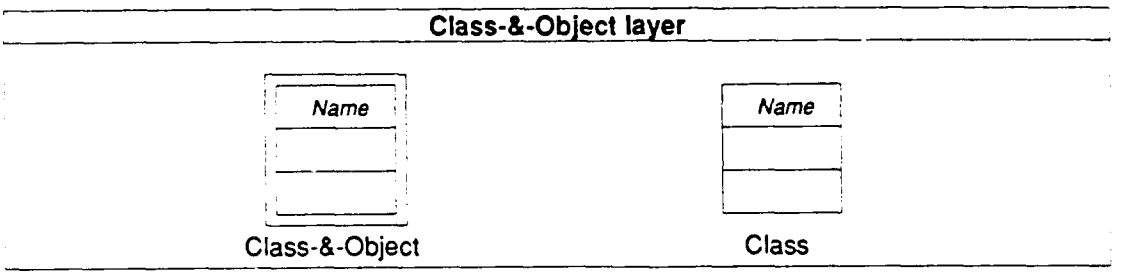

Figure F-4. Class-&-object layer

The class-&-objects are represented by the 'Class-&-Object' symbol (see left symbol in figure F-4). A variation on the 'Class-&-Object' symbol is the 'Class' symbol (see right symbol in figure F-4). This symbol is used to represent a generalization class from the problem domain, whose corresponding objects are portrayed by its specializations which have Class-&-Object symbols. The class-&-objects and classes form the Class-&-Object layer.

Identifying structures. In this activity you have to find the structures. Structure is an expression of problem-domain complexity, pertinent to the system's responsibilities. The term structure is used as an overall term, describing both generalization-specialization structure and whole-part structure.

The generalization-specialization (Gen-Spec) structure may be viewed as part of the distinguishing between classes aspect. From the specialization's perspective a Gen-Spec structure can be thought of as an 'is a' or 'is a kind of' structure.
The notation for the Gen-Spec structure is shown in figure F-5. A semi-circle marking with connecting lines. The top line is connected with the generalization class; the lines below point to the specialization classes.

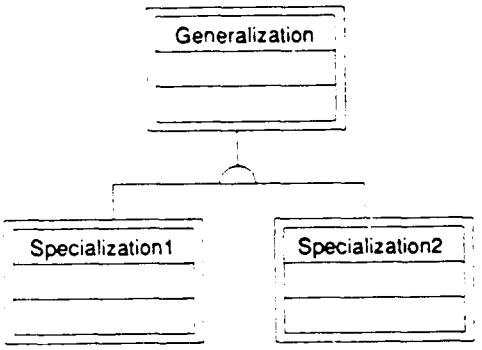

Figure F-5. Gen-Spec structure notation

Besides the Gen-Spec structure there is the Whole-Part structure. This structure groups class-&-objects together based upon the whole-part meaning. The notation is shown in figure F-6 with a whole object (of a class-&-object) at the top and a part object (of a class-&-object) below.

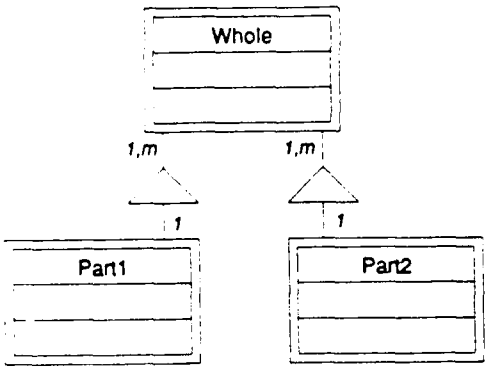

Figure F-6. Whole-Part structure notation

Each end of a Whole-Part structure line is marked with an amount or range, indicating the number of parts that a whole may have and vice versa.

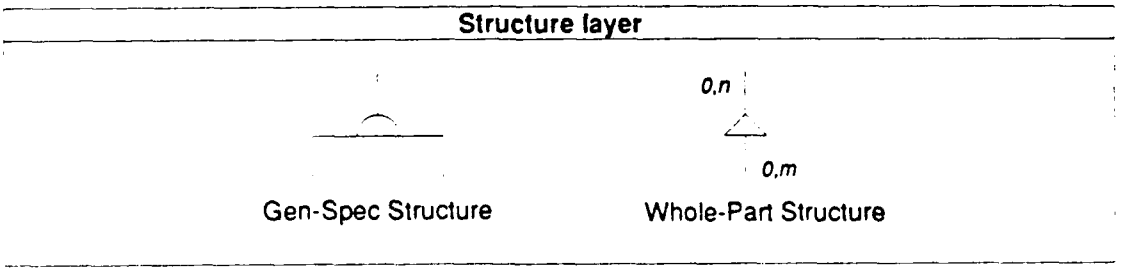

Figure F-7. Structure layer

The two structures form the Structure layer of the OOA model (see figure F-7).

Identifying subjects. A subject is a mechanism for guiding a reader through a large, complex model. Subjects are also helpful for organizing work packages on larger projects, based upon initial OOA investigations. Subjects facilitate communication and avoid information overload.

Subjects can be shown in three forms: collapsed, partially expanded or expanded (see: figure F-8). The partially expanded subject just lists its class-&-objects. The expanded subject is a rectangular box around the class-&-objects which are included.

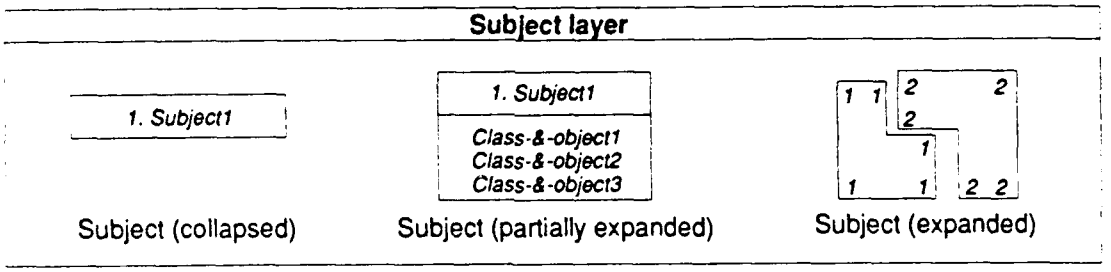

Figure F-8. Subject layer

The best way to identify the subjects is to promote the uppermost class in each structure and each class-&-object not in a structure upwards to a subject. Then refine the subjects by using minimal interdependencies and minimal interactions between them.

Defining attributes. In this activity the attributes have to be defined. An attribute is some data (state information) for which each object in a class has its own value. The way to define the attributes is to identify them; position the attributes; identify Instance connections; check for special cases; and specify the attributes. Instance connection is a model of problem domain mapping(s) that one object needs with other objects, in order to fulfil its responsibilities.

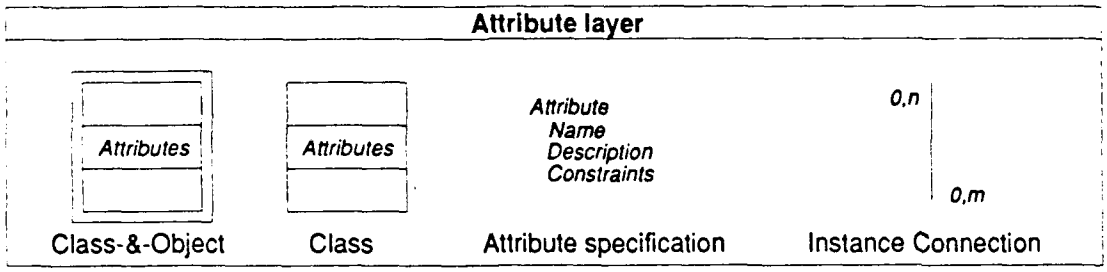

Figure F-9. Attribute layer

The attributes and their specifications and the instance connection form the Attribute layer (see figure F-9).

Defining services. In this activity the services are defined and the five layer OOA model is set together. A services is a specific behaviour that an object is responsible for exhibiting. The strategy to define the services is: identify object states, identify the required services; identify message connections; specify the services; and put the OOA documentation together. The class-&-object specifications with object state diagrams, additional constraints and services, is the lowest layer of the OOA model (see figure F-10).

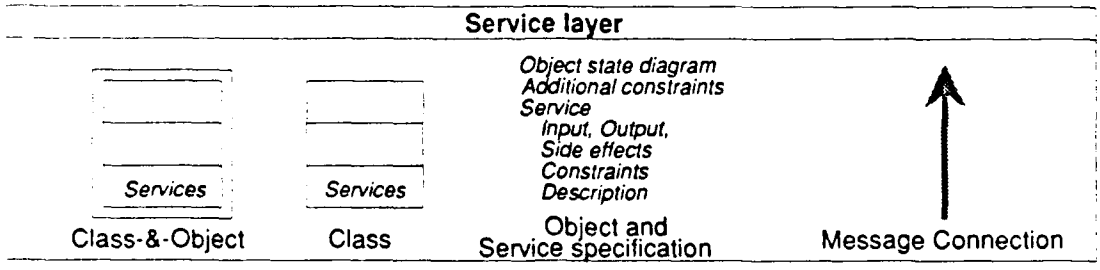

Figure F-10. Service layer

As a result of these five layers the OOA model is completed. The model includes its notation symbols and the class-&-object specifications. Each class-&-object has a specification according to the following template:

(3) SourceAndType

(4) Display

The four subjects of the object-oriented analysis model can graphically be represented by a box (see figure F-13).

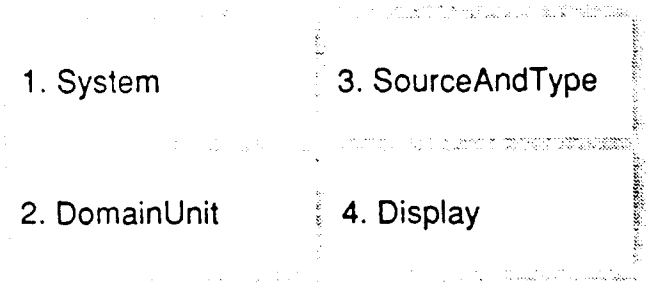

Figure F-13. Subjects (collapsed)

If we want to look at these subjects in more detail we can expand the subjects in the model. In this section we just look at the top three layers (Subject layer, Class-&-object layer and Structure layer) so we won't see the attributes and the services.

System subject. First, we are going to take a look at the System subject. The following class-&-objects are in the System subject:

System

Project

Domain

NodeTreeShell (a class)

In the next figure (F-14) the System subject is expanded for the top three layers.

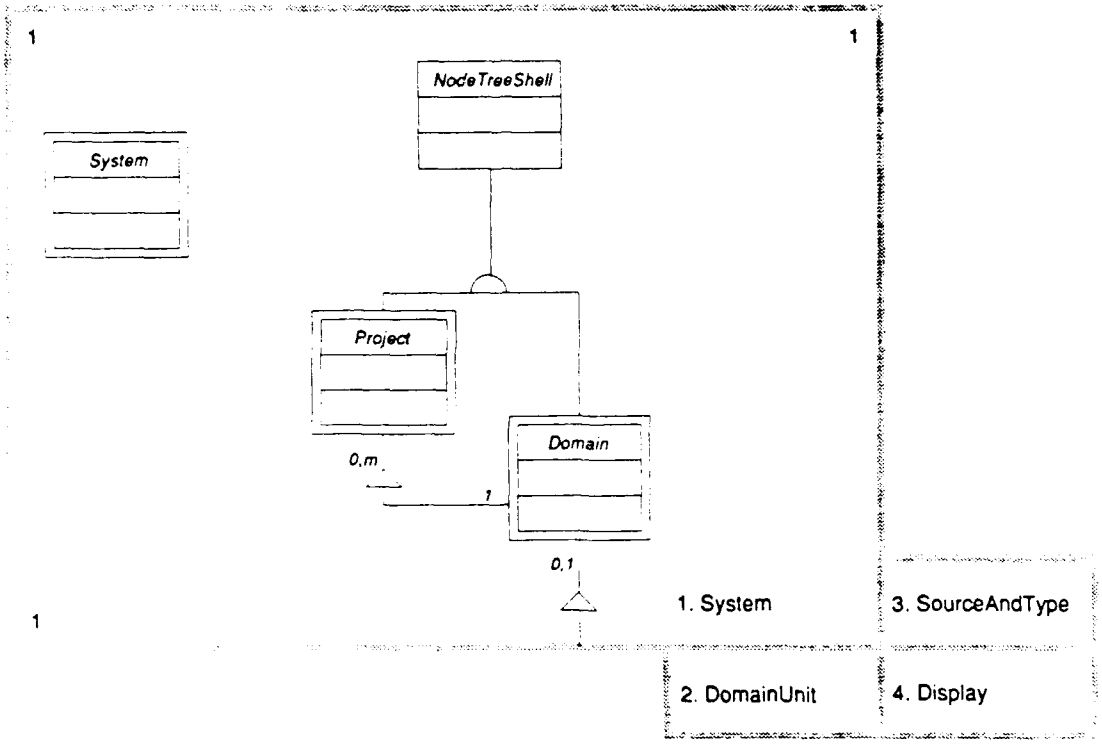

Figure F-14. System subject (expanded)

DomainUnit subject. The next subject DomainUnit contains the following class-&-objects:

SimpleUnit
Link
HorizontalUnit
Node
Group
VirtualNode
VerticalUnit
InformationCollection (a class)

In the next figure (F-15) the DomainUnit subject is expanded for the top three layers.

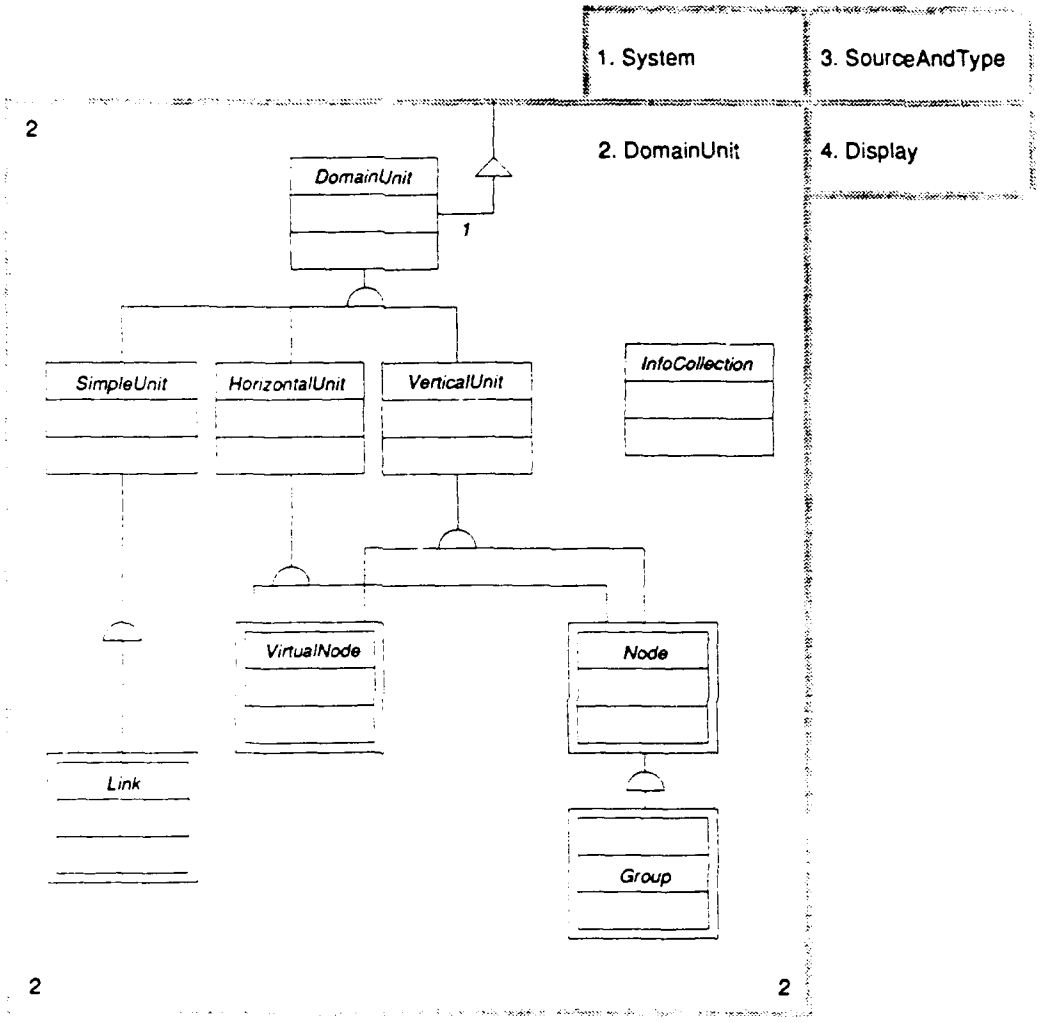

Figure F-15. DomainUnit subject (expanded)

SourceAndType subject. SourceAndType is the next subject we are going to observe. It contains the following class-&-objects:

Source (a class)

AuthorSource

ReferenceSource

PersonReference, BookReference and OtherReference

Type

The next figure F-16 shows the SourceAndType subject expanded for the top three layers.

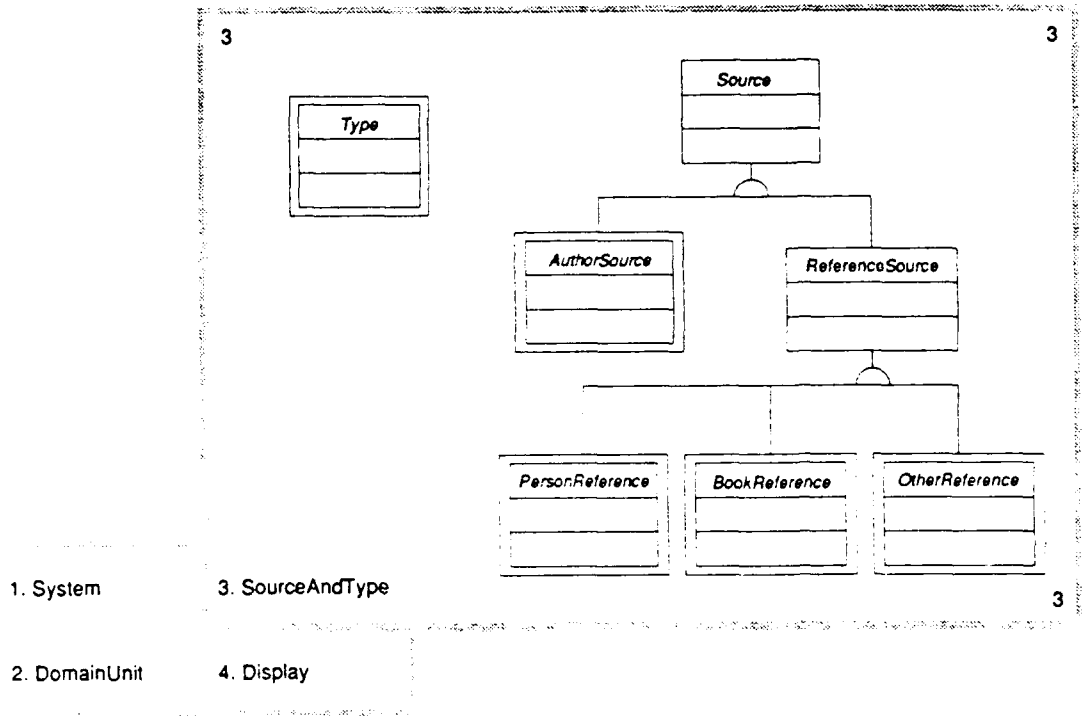

Figure F-16. SourceAndType subject (expanded)

Display subject. The last subject Display contains various class-&-objects related to the display, like: Rectangle, Box, Menu, Pane, Button, GraphicPane, ListPane, Text, ScrollBar, Window, TextWindow, DialogWindow, GraphicWindow, Line and ArrowHead. Most of these classes are standard, and so already defined, display elements. The ones that are not defined are the following (with their generalization class between the parentheses):

NodeBox (Box), VirtualNode (Box), GroupBorder (Box), AttributePopUpWindow (PopUpDialogWindow), NodeBrowserWindow (PopUpDialogWindow), GraphicNodeBrowser (GraphicWindow), NodeWindow (GraphicWindow), IconWindow (GraphicWindow), and Arrowhead (GraphicObject)

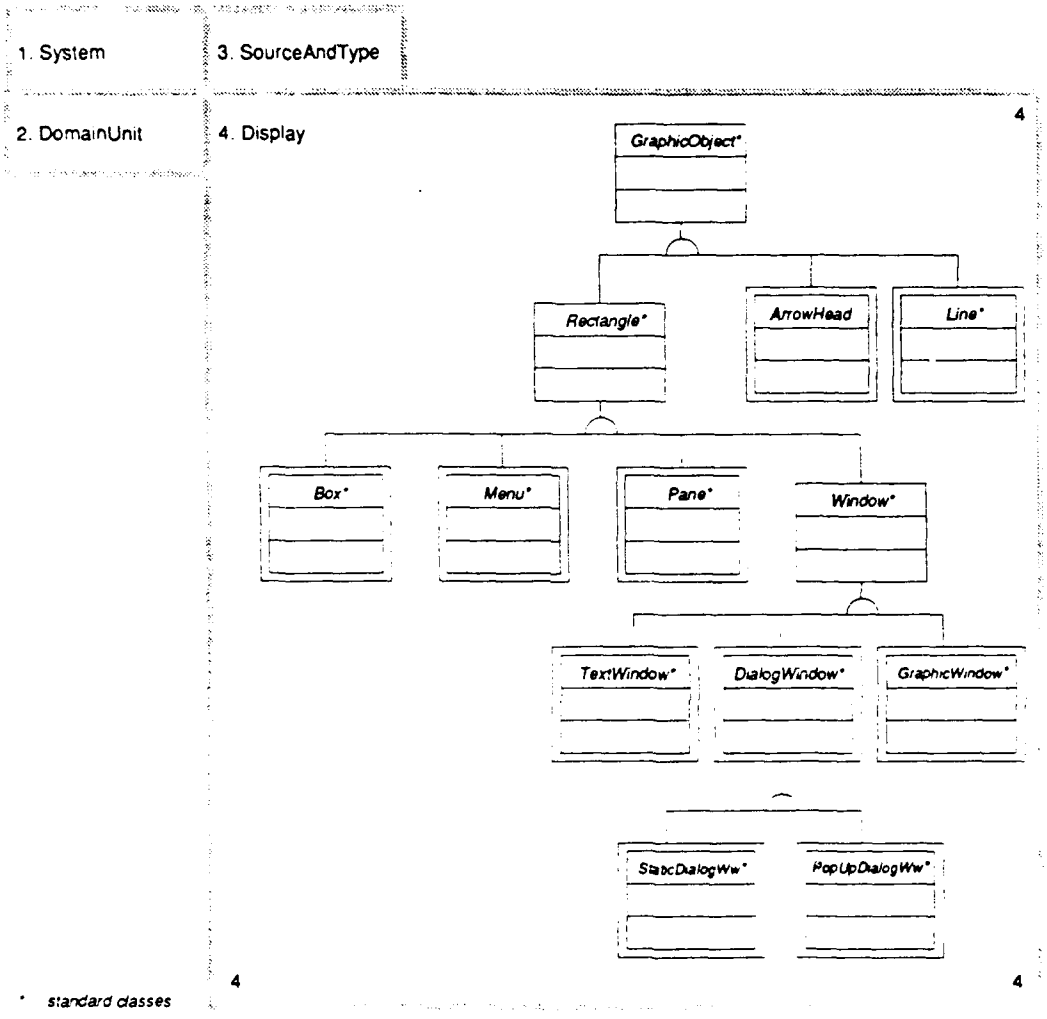

Figure F-17. Display subject (expanded)

In the figure F-17 the Display subject is expanded for the top three layers. However not all classes and class-&-objects in this subject are presented.

5 CONCLUSION

In this paper we have presented an object-oriented analysis for a knowledge organization system called Organon. This analysis covers the major classes and objects that would be present in a first draft of the system. The analysis can be used to (1) develop a detail design for the system, or (2) if time is critical, it could even be used as a template to start coding. One advantage of the method used to do this analysis is that it is fairly language independent, so that a second version of Organon could be coded based on this analysis in Lisp/CLOS, C++, or SmallTalk with few, if any, analysis level changes.

Unfortunately, the analysis for our current conception of Organon is incomplete. This is due primarily to time constraints and the size of the system envisioned. A second paper, now in preparation, will address this problem to some extent. This paper will present the essential message structure for the Organon system and will provide a brief look at the system from the user's perspective.

This analysis should not be considered the final version of Organon; rather it is a snapshot of an evolving system. Futures features of the system now under consideration include the addition of a schema/case recognizer that will enable the system to interact in an intelligent way with the user, and the extension of the Organon system so that it is capable of handling multiple users in a real-time collaborative environment.

6 REFERENCES

Arend, J. G. M. van der, and Brooks, L. W. (in preparation) Organon : A tool for graphical knowledge organization. Version 2: Object-Oriented Analysis II (Messages and User Perspective). ARI Working paper, U.S. Army Research Institute, Alexandria, VA, USA.

Brooks, L. W. (in press). Organon : A tool for graphical knowledge organization. Version 1: Introduction and description. ARI Working paper, U.S. Army Research Institute, Alexandria, VA, USA.

Brooks, L. W., and Jardine, C. R. (in press). Organon : A tool for graphical knowledge organization. Version 1: Source Code and Commentary. ARI Working paper, U.S. Army Research Institute, Alexandria, VA, USA.

Coad, P., and Yourdon, E. (1991). Object-oriented analysis. Englewood Cliffs, New Jersey: Yourdon Press.

Cox, B. (1986). Object oriented programming. Reading, Mass.: Addison-Wesley.

Goldberg, A., and Robson, D. (1983). SmallTalk-80: The language and its implementation. Reading, Mass.: Addison-Wesley.

Keene, S. (1988). Object-oriented programming in common lisp. Reading, Mass.: Addison-Wesley.

Stroustrup, B. (1986). The C++ programming language. Reading, Mass.: Addison-Wesley.

Xerox. (1985). NoteCards. Release 1.2i. Reference Manual. Xerox Palo Alto Research Center, California, USA.

Appendix G:

THE ORGANON PROTOTYPE

ORGANON: A TOOL FOR GRAPHICAL KNOWLEDGE ORGANIZATION

Version 2: Prototype

J. G. M. van der Arend
TNO Physics and Electronics Laboratory at The Hague,
The Netherlands

U.S. ARMY RESEARCH INSTITUTE
5001 Eisenhouwer Avenue, Alexandria, Virginia 22333-5600

June 1991

ORGANON: A TOOL FOR GRAPHICAL KNOWLEDGE ORGANIZATION. VERSION 2.0 PROTOTYPE

CONTENTS

1 INTRODUCTION

This documentation describes the Organon version 2 Prototype on the NeXT computer.

On the NeXT computer there are actually two version 2 prototypes. Version 2.0 is the prototype for the Object-Oriented Analysis (OOA); version 2.1 is the prototype for the Object-Oriented Design (OOD). This last version is described in this document. This document will use the term Organon version 2 Prototype for the most recent version: Organon version 2.1.

2 ORGANON VERSION 2 PROTOTYPE

The Organon version 2 is a result of the Organon version 1 (see Brooks) and the efforts of Larry Brooks (U.S. Army Research Institute) and Jos van der Arend (TNO Physics and Electronics Laboratory) during the period March till September 1991. These efforts where previously described in a working paper (see van der Arend and Brooks).

The Organon version 2 Prototype was built with the Interface Builder and Objective-C. The Interface Builder is described in chapter 8 of 'The NeXT System Reference Manual'; Objective-C and object-oriented programming on the NeXT is described in chapter 3 of the same manual.

The example knowledge in this prototype is Mission Training Plan of the ARTEP (Army Training and Evaluation Program). The knowledge as presented in the prototype is described in a concept study of the Computer-Aided ARTEP Production System (see Bloedorn et al.)

2.1 Source Code

The source code for the prototype consists of two main parts: the source code for the Interface Builder and the Objective-C code. Both source codes are required to make the executable code of the application.

The source code for the Interface Builder includes the following files:

'Domain.nib', 'DomainIcon.tiff', 'IB.proj', 'Makefile', 'NodeIcon.tiff', 'Organon.iconheader', 'Organon.nib', 'OrganonIcon.tiff', 'Organon_main.m' and 'ProjectIcon.tiff'.

Here is a short description of each file:

IB.proj (the Interface Builder project file);

Organon.nib (the Interface Builder interface file for the main Organon module);

Domain.nib (the Interface Builder interface file for the Domain module);

Makefile (specifies which files are required by the compiler and linker to build the application);

Organon_main.m (contains the main() C function);

Organon.iconheader (contains information which the Workspace Manager uses to associate icons with the application);

DomainIcon.tiff, NodeIcon.tiff, OrganonIcon.tiff, and ProjectIcon.tiff (icon files).

The Objective-C source code includes the class definition of the object classes: the class interface and class implementation files. The class interface file (file with file name class name and extension ".h") declares the interface to the class; the class implementation file (file with file name class name and extension ".m") actually defines the class.

The Objective-C source code files are:

DialogManager.[hm]

Domain.[hm]

DomainUnit.[hm]

Group.[hm]

HelpPanel.[hm]

HelpSystem.[hm]

HorVertUnit.[hm]

InfoCollection.[hm]

Inspector.[hm]

InspectorPanel.[hm]

Link.[hm]

Node.[hm]

NodeBrowser.[hm]

NodeWindow.[hm]

Project.[hm]

SimpleUnit.[hm]

System.[hm]
VirtualNode.[hm]
('ClassName.[hm]' means 'ClassName.h' and 'ClassName.m')

These files contain the class definition for the object classes.

Some of the ".h" files, the ones not starting with a class name, are included in the definition files of a class implementation file. These are the file 'domainMTPdev.h', 'domainOthers.h', 'globals.h' and 'helpText.h'.

- domainMTPdev.h and domainOthers.h are include files for the class definition file 'Domain.m'. They are used to divide this implementation file into smaller files. The first file defines the domain 'MTPdevelopment', the second the other domains: 'ARTEP', 'ASAT', 'MTPdescription' and 'MTPhistory'.
- helpTexts.h is an include file for the class definition file 'HelpPanel.m'. This file describes the help texts for the Help Panel.
- globals.h is an include file for the definition of global variables.

2.2 The Executable Prototype

The executable code for the prototype consists of one file 'Organon.debug' in directory '/me/Jos/Organon.v2.1/ProtoIB'.

The executable code file is generated by the Interface Builder. One way to make this executable is:

1. Start Interface Builder:
   - Double click on Interface Builder's icon in the Dock, if there, or start Interface Builder from the Workspace Manager from its location in directory '/NextApps'.
2. Open NeXT Interface Builder file:
   - Click on the 'Open...' command in the File Menu.
   - Browse through the directory structure in the Open Panel to the directory '/me/Jos/Organon.v2.1/ProtoIB'.
   - Select file 'Organon.nib' (or 'IB.proj') and press the 'OK' button in the Open Panel.
3. Start building the executable version:
   - Click on the 'Make' command in the File Menu.

4 REFERENCES

Arend, J. G. M. van der, and Brooks, L. W. (in press). Organon : A tool for graphical knowledge organization. Version 2: Object-Oriented Analysis I (Classes and Objects). ARI Working paper, U.S. Army Research Institute, Alexandria, VA, USA.

Bloedorn, G.W., W.H. Crooks, M.D. Merill, H.J. Saal, L.L. Meliza and O.I. Kahn (1985). Concept Study of the Computer-Aided ARTEP Production System (CAPS). ARI Research Report 1403, U.S. Army Research Institute, Alexandria, VA, USA.

Brooks, L. W. (in press). Organon : A tool for graphical knowledge organization. Version 1: Introduction and description. ARI Working paper, U.S. Army Research Institute, Alexandria, VA, USA.

Larkin, Don, et al. (1989). The NeXT System Reference Manual. Release 1.0 On-line Edition, NeXT, Inc., Redwood City, CA, USA.

Appendix H:

REPORT OF DOD'S T2TG MEETING

4th DOD Training Technology Technical Group (T2TG) Meeting,

Arlington, 19-20 March 1991

U.S. Army Research Institute

Alexandria, Virginia

Jos van der Arend

March 21, 1991.

This is a report of the presentations on the T2TG meeting. Overall there were about 80 persons from the DoD present at the meeting. The program of this meeting is presented in figure G-1.

PLENARY SESSION I
Dr. Seidel: Introduction of speakers
Dr. Robert Gagne, Senior Research Associate, National Research Council Air Force Armstrong Laboratory
Dr. Andrew Mollnar, Advanced Applications for Technology, NSF
Dr. Judith Orasanu, Principal Investigator, Aerospace Human Factor Research Division, NASA Ames Research Division

SUBGROUP SESSIONS
CREW GROUP TEAM AND UNIT TRAINING
Aircrew coordination training:
- Introduction NASA (Kanki)
- Introduction Army (Leedom)
- Introduction Air Force (Nullmyer)
- Automated instructor support for aircrew coordination training (Hamburger)
- Joint service proposal (Engelhardt)

Measuring tactical performance:
- Introduction (Salas)
- Team process measures (Canon-Bowers)
- Measures of tactical team training effectiveness (Dwyer)
- Evaluating training effectiveness (McDaniel)

TRAINING DESIGN AND EVALUATION
(information exchange on research activities)

ADVANCED TRAINING TECHNOLOGIES
- FIS: an expert system for fault isolation (DeJong)
- The use of a mental model to teach problem solving in electronic trouble-shooting (Park)
- Visual problem solving (Psotka)
- Cognitive modelling for skill acquisition of Morse code (Wisher)
- Teaching of literacy skills and problem solving skills (Steuck)

PLENARY SESSION II
Report of subgroup sessions
Dr. Seidel: Summary and future look

Figure G-1 Program of the T2TG meeting

The sequence of this report is according to the sequence of the presentations in the agenda (see figure G-1). The first three were speakers in the plenary session (Tuesday morning); the other speakers presented their work in the Advanced Training Technologies subgroup sessions

(Tuesday afternoon and Wednesday morning). I did not attend the other subgroup sessions on crew group team and unit training and training design and evaluation.

The overhead sheets of all speakers will be available later.

Dr. Robert Gagne (USAF),
talked about the learning processes in instruction and the instructional aim of these processes and the means to establish this aim:

	Learning Processes	Instructional Aim	Means
1	expectancy	communicate goal	verbal statement; picture; demo
2	perception	distinguish features	diagram; arrows; bold print
3	working storage	limit the items	chunking; imaging; automatic processing
4	composing	optimize meaning	embedded in text; table; diagrams
5	storing	practice; elaboration	related communication; spaced reviews
6	retrieval	cue to familiar	well-known sound, rhyme; meaning
7	generalization	common elements; mindful abstraction	productions; relate to conditions of use
8	gratifying	provide feedback	confirming the expectancy

Figure G-2 Overview Gagne's theory

Dr. Andrew Molnar (NSF),
talked about his general impressions of instruction in the US because of the information explosion and the change of instruction:

"to know has changed in to know how to access the information"

He saw the following applications for advanced technology:

1 KBS and ITS
2 intelligent tools
3 production and authoring systems
4 problem-solving and programming
5 strategic projects

Dr. Judith Orasanu (NASA),

presented the concepts of 'team decision making'. She explained the terms in team decision making (TDM), the characteristics and ways how to improve the decision.

Influences on the best decision are: participants, information, communication and decision strategy. After the decision follows the action. The decision and action are influenced by coordination and situation cues. The action influences the goal what influences the decision.

The elements of TDM are:

- shared mental models
- effective communication
- good resource management
- task-appropriate decision strategy

Options to improve TDM are: training, decision aids and organizational structure.

Dr. Ken DeJong (GMU, Navy),

presented the FIS: an AI-based fault-isolation system. The issue is there are a number of replaceable modules and the user must try to find the fault module by doing one or more tests for every module. As a result of these tests the user can decrease the number in the set of possible fault modules until the number is one.

The system has qualitative causal modeling, a probabilistic model of beliefs, intelligent test and replacement recommendations, and a powerful KA strategy. His biggest issues was: "how to turn this system into a training system".

Dr. Ok-choon Park (ARI),

presented his work about the use of a mental model to teach problem solving in electronic trouble-shooting (ET). They build a lesson about ET with 3 parameters:

1 graphical displays (static or dynamic)
2 instruction context (with or without context)
3 instruction sequence (whole-part or part-whole)

An evaluation session had the following sequence: questionnaire (QN), instruction, QN, practice, QN, training, QN.

The evaluation sessions with 96 students, divided over the six groups, are just finished. No conclusions yet.

Dr. Joe Psotka (ARI),

presented the project 'Multimedia aircraft recognition trainer'. The trainer is a hypercard-based information retrieval system on a Mac. The information about every plane is only one screen containing: graphical image of fore, top and side view and the textual information/description like: country of origin, similar aircraft, crew, role, dimensions, wings, engine(s), fuselage, tail.
The user can get the information in the database in various ways. The user can also compare the images of the planes (side-by-side or on top of each other) or make 2 sets (friend and foe) of planes. The main components of the trainer are:

- search tools
- main planes description
- weft (wings, engine, fuselage, tail)
- student info

The various uses of the tool can be examined and the user can place the images of the aircraft in a 2D space to show their relations.
Future extensions could be: 3D view of aircraft, add video scenes and build some intelligence.

Dr. Bob Wisher (ARI),

told about his Morse code project. In this project they try to better understand the acquisition and retention of cognitive skills. They build a model of the acquisition of cognitive skills and their retention for Morse code and generalize this. The stages in the cognitive process are: auditory reception, character recognition, motor organization and response execution. They have a mathematical model now.
His future work might be to specify a similar cognitive model for the gunnery tank crew.

Dr. Kurt Steuck (Defense ALHRT),

presented the management and research work in the fundamental skills training (FST) project.
Together with industry and government his team is developing ITSs for mathematics, English writing and science for high school students on a Mac. The architecture of one of these systems contains the modules:

- problem DB
- core curriculum DB
- problem selection rules
- instructional DB

- student model
- interface
- control rules
- update rules

The first ITS (math) will be ready September 1991. Then they are going to use this system for the entire schoolyear and make revisions. The revised ITS will be ready for the next schoolyear (September 1992). The other ITSs will evolve in the same way but exactly 1 (writing) and 2 (science) years later.

The instructional approaches can be varied between:

- guided discovery
- reflective teaching
- opportunistic teaching
- direct instruction (instructor controlled)

Dr. Ray Perez (ARI)

presented his work on building a mental model for developing training. The model will be based on the interviews the project team had with four training development experts. Interviews include questions about best, toughest, worst training system and solving a problem. The project team is now going to look at the data to formulate the model.

Based on this work, a set of tools for the developer of training/instruction is going to be developed. Organon, the system I am currently working on, will be one of those tools.

Appendix I:

ID EXPERT

ID Expert v3.0

Expert System for Instructional Design

U.S. Army Research Institute

Alexandria, Virginia

Jos van der Arend

April 24, 1991

CONTENTS

1 INTRODUCTION

"The Department of Defense is significantly involved in the development of computer-assisted instruction (CAI). Experience with CAI has confirmed that design is the most expensive part of CAI development, entails the greatest risk, and requires the highly specialized skills of instructional designers. DoD wishes to use expert system technology to create an electronic job aid to support the design process, thus enabling lower skilled designers to efficiently design effective instruction...
The purpose of this project is to further develop a prototype job aid to guide instructional design. This job aid is titled the Instructional Design Expert System (ID Expert)."

This report is about this ID Expert, particularly version 3.0. The text above comes from the final report of the implementation of ID Expert version 3.0 (See: Jones et al.., April 1991). This report is the foundation of this short report. Besides this report I read various publications from the list of references and attended a demonstration given by Human Technology Inc.

ID Expert is based on the theory of David Merrill at Utah State University. The first generation of Merrill's work is called the Component Display Theory (CDT). The CDT was first published in 1987 (See: Merrill, 1987). The second generation of the instructional design theory (ID2) was developed by Merrill, Li and Jones (See: Merrill, Li and Jones, February 1990).
The development of ID Expert v3.0 was sponsored by the U.S. Department of Defense, in cooperation with the U.S. Office of Personnel Management and Human Technology Inc.

ID Expert v1.0 was implemented on a VAX computer using the expert system shell S.1. Version 2.0, like version 1.0 funded by the Army Research Institute for the Behavioral and Social Sciences, was implemented on a desktop platform using Macintosh computers and the expert system shell Nexpert with a Hypercard interface. An interactive graphic user interface was constructed to navigate through the instructional design process.
Version 3.0 is implemented using the prototyping system ToolBook and Windows 3.0, on IBM compatible 386 computers. The outcome was the integration of all three basic components of ID Expert:

Knowledge Analysis Component (KAC),
Strategy Analysis Component (SAC), and
Transaction Shell Component (TSC).

The ultimate goal of the ID Expert research is to develop an integrated platform to support all phases of the instructional development process: analysis, design, development, delivery and evaluation. The initial focus is on the first three phases.

2 ID1 AND ID2

The research team itself come with a number of limitations of the current first generation of instructional design (ID1) theories:

1. Content analysis does not use integrated wholes which are essential for understanding complex and dynamic phenomena.
2. Limited prescriptions for knowledge acquisition.
3. Limited prescriptions for course organization.
4. Existing theories are essentially closed systems.
5. *ID1 fails to integrate the phases of instructional development.*
6. ID1 teaches pieces but not integrated wholes.
7. Instruction is often passive rather than interactive.
8. Every presentation must be constructed from small components.
9. Current ID1 is labour intensive.

To overcome these limitations the research team is expanding the instructional design theory itself. The ID2 would address the above limitations. ID2 is:

- capable of analysing, representing and guiding instruction to teach integrated sets of knowledge and skills;
- capable of producing pedagogic prescriptions for the selection of instructional strategies and the selection and sequencing of instructional transactions;
- an open system, able to incorporate new knowledge about teaching and learning and to apply these in the design process;
- able to integrate the phases of instructional development.

ID2 comprises the following components:

1 a theoretical base that organizes knowledge and defines a methodology;
2 a means of representing domain knowledge for making instructional decisions;
3 a collection of mini-experts for a particular ID decision or set of those;
4 a library of instructional transactions for the delivery of instruction, and the capacity to add new or existing transactions to the library;

5 an online intelligent adviser program that dynamically customizes the instruction during delivery.

3 ID EXPERT VERSION 3.0

ID Expert has three principal tools/components, which have been implemented in version 3.0:

Knowledge Analysis Component (KAC),

Strategy Analysis Component (SAC), and

Transaction Shell Component (TSC).

Eventually, a variety of tools will be developed for ID Expert. The tools are independent, but share data. Each tools directly communicates with the user in order to carry out its functions.

The KAC places the results of its analysis into the domain knowledge base. Each transaction shell "knows" what knowledge it can use. The SAC uses its own rule base, augmented by the rule bases *of each shell, to prescribe* transactions for authoring. The student and environment characteristics are used by the transactions to customize their functioning. See figure I-1.

3.1 Knowledge Analysis Component (KAC)

The KAC aids the subject matter expert to represent the domain to be instructed in a knowledge representation. It incorporates the following functionality:

- represent entities (devices, objects, persons, creatures, places, symbols), and the activities (action in which a learner plays a role);
- represent parts of entities and steps of activities;
- represent abstractions (kinds of activities and steps of activities);
- represent associations (network links among entities and activities);
- store all knowledge;
- support interactive navigation through the knowledge base by user control.

KAC represents knowledge by objects they called frames; each frame has an internal structure and external links to other frames. A set of frames linked together is called a network. There are three kind of frames defined: entities, activities and processes.

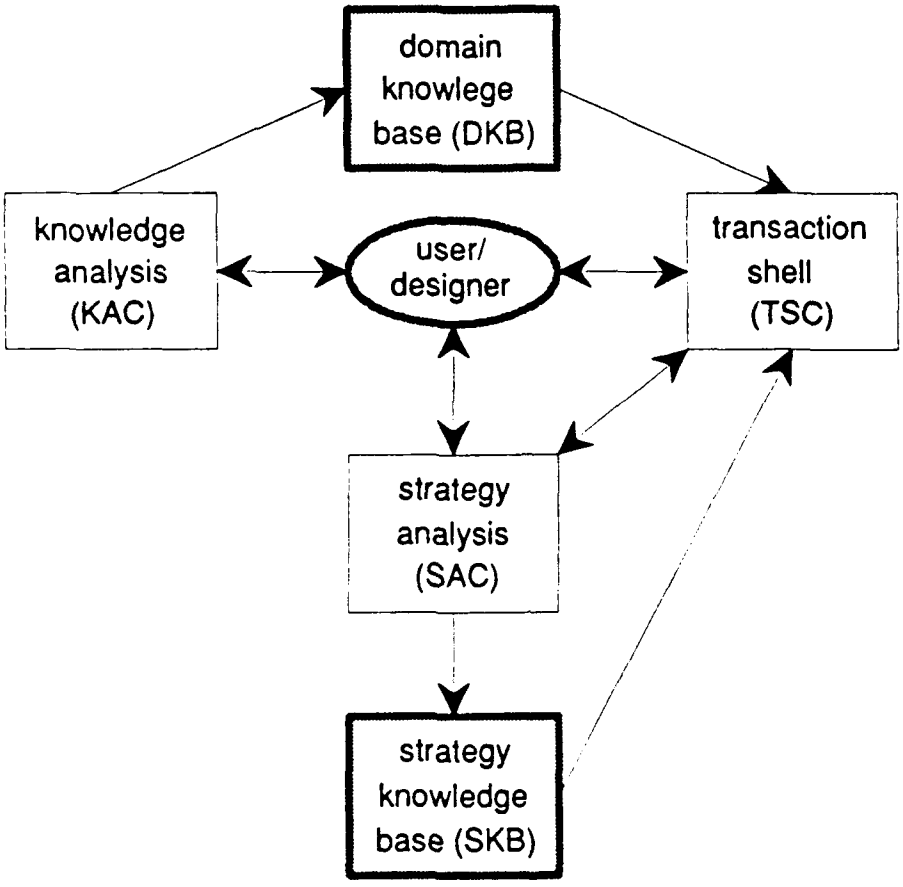

Figure I-1. Interactions of SAC with KAC and TSC.

3.2 Transaction Shell Component (TSC)

Transaction shells are instructional components which implement a transaction of a particular class. The shells in the TSC exhibit the following functionality:

- implement an instructional algorithm for a given type of control;
- incorporate parameters to vary the action of the transaction;
- recommend default values for all parameters;
- circumstances under which that shell should be described.

The parameters contained in version 3.0 are: focus, content, coverage, view, trials, mastery level, response mode, modes, strategy, strategic control and tactical control. Other parameters are: guidance level, guidance type, vertical sequence, temporal sequence, feedback, replacement, items, time out, and item order.

The current ID Expert version is capable of authoring three transaction shells (Identify, Execute and Classify), and delivery of one shell (Identify).

3.3 Strategy Analysis Component (SAC)

The SAC integrates the KAC and TSC to create instruction. SAC is the means by which the author identifies the enterprise (higher level tasks which require an integrated set of knowledge and skills) to be instructed, selects content for each enterprises, selects higher level strategies for sequencing of instruction, and customizes the individual transactions. It:

- gathers generally applicable information about the audience and resources and constraints of the setting (course, student and instructional environment) and stores this in the strategy knowledge base (SKB).
- recommends instructional transactions to instruct content identified by the KAC;
- directs and constrains further knowledge acquisition based on recommended transactions;
- prioritize authoring of transactions.

So, the SAC demonstrates the following functionality:

- select content on instruction;
- select overall instructional strategy;
- prescribe transactions for content;
- sequence transaction;
- account for student and environmental attributes
- apply expertise to the prescriptions;
- demonstrate a rapid prototype approach to design;
- demonstrate interactions of SAC with KAC and TSC.

3.4 User Interface

The description of the interface is illustrated by the screens images.

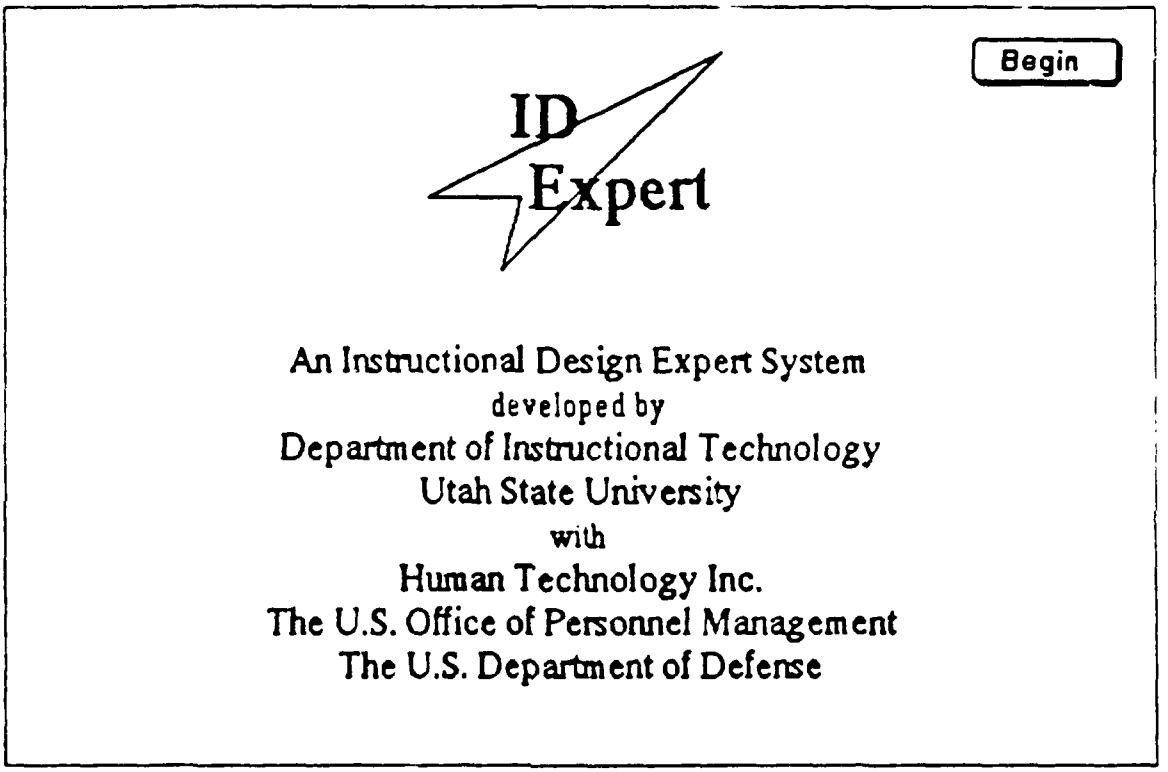

Figure I-2. ID Expert's title screen

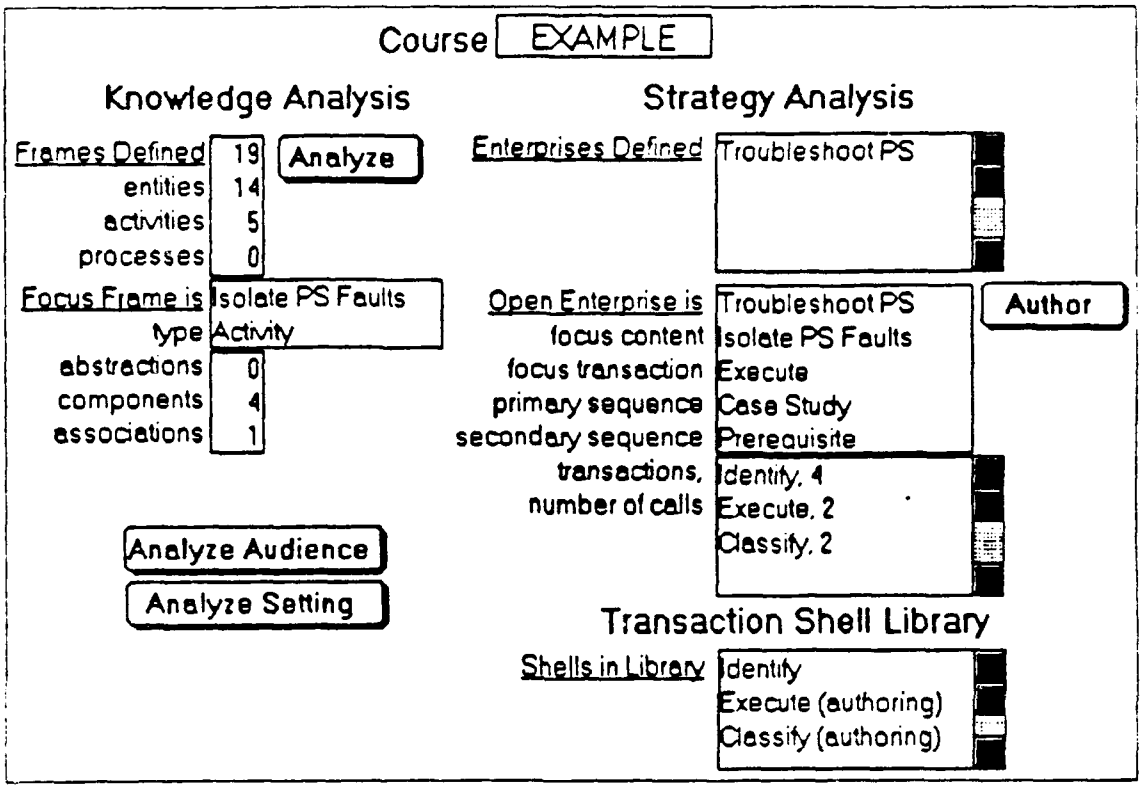

Figure I-3. ID Expert's Command console

After the title screen (figure I-2) the user comes into the ID Expert Command console (figure I-3). From here the user enters one of the components of the system. However, the user will always return to this screen before entering other components.

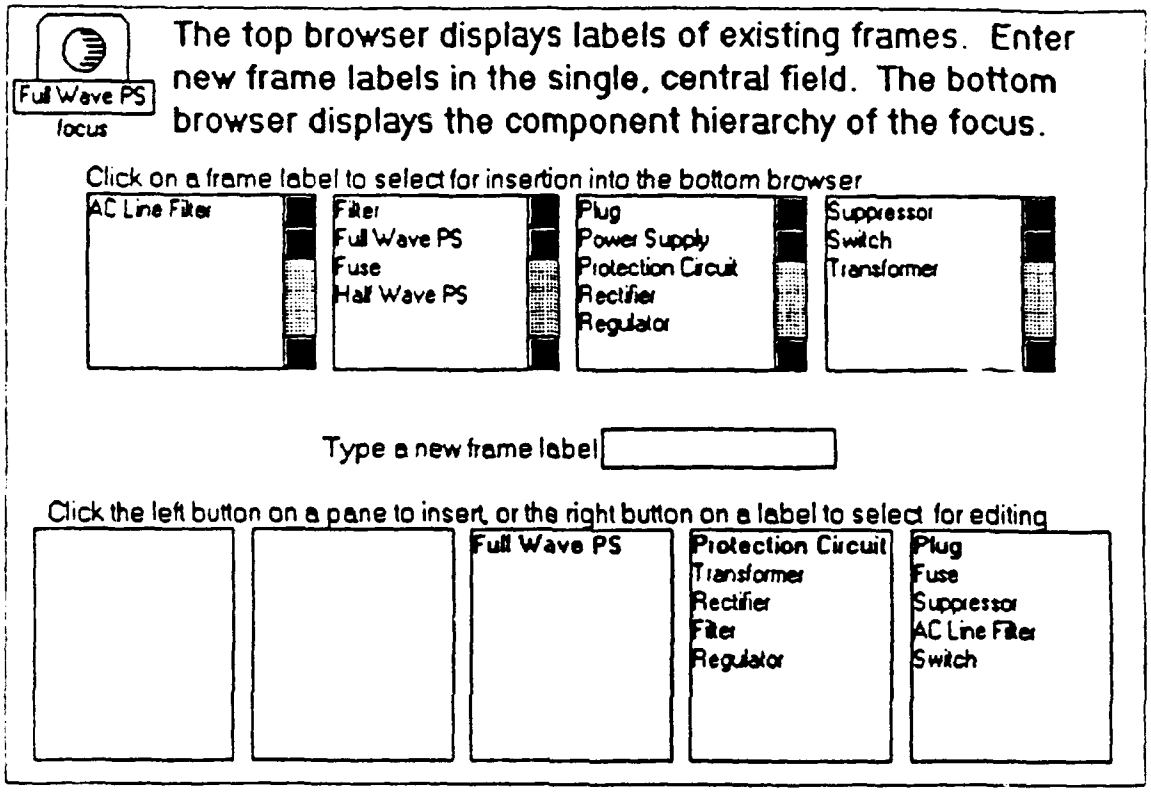

Figure I-4. ID Expert's Component hierarchy browser

Under the KAC there are 4 general browsers (abstraction hierarchy, components hierarchy, property and association) which all look-alike (figure I-4). The top of the display is the browser for existing frames in the knowledge base. The panes to the right of the focus indicates subclasses, and the panes to the left of the focus indicates superclasses. The bottom of the display is a browser to edit the focus frame.

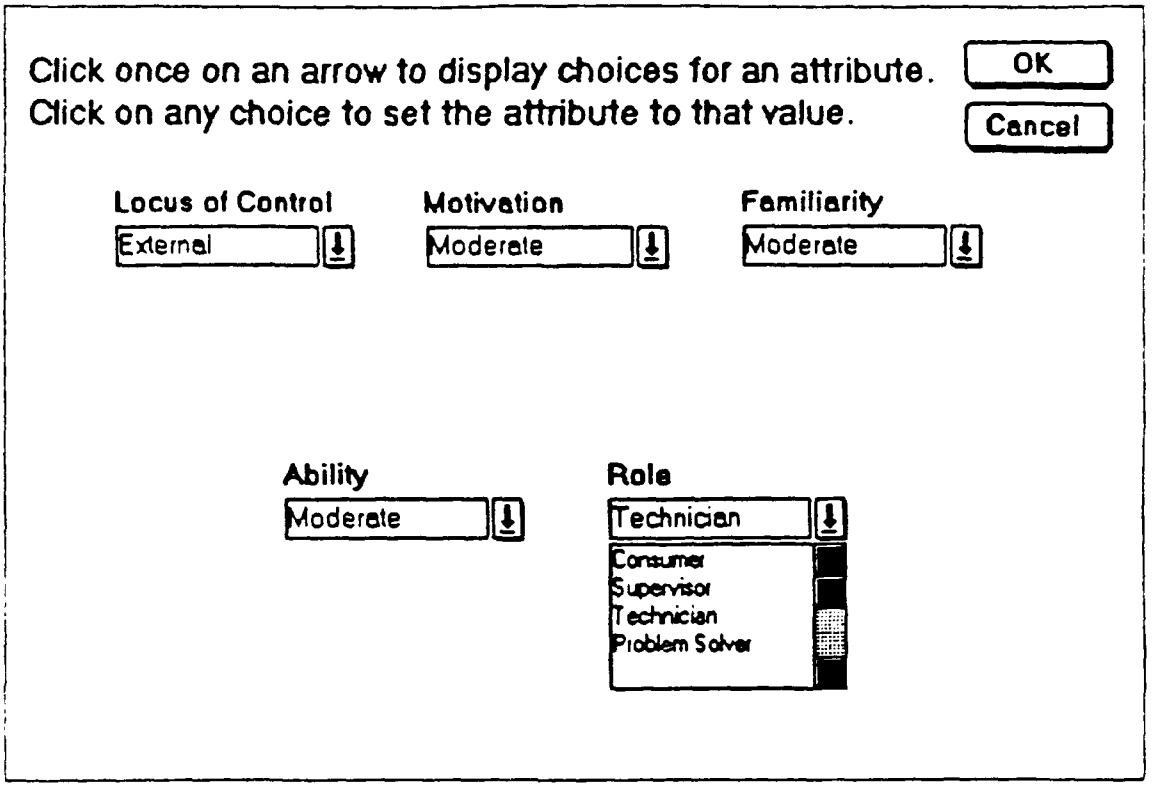

Figure I-5. ID Expert's audience attributes

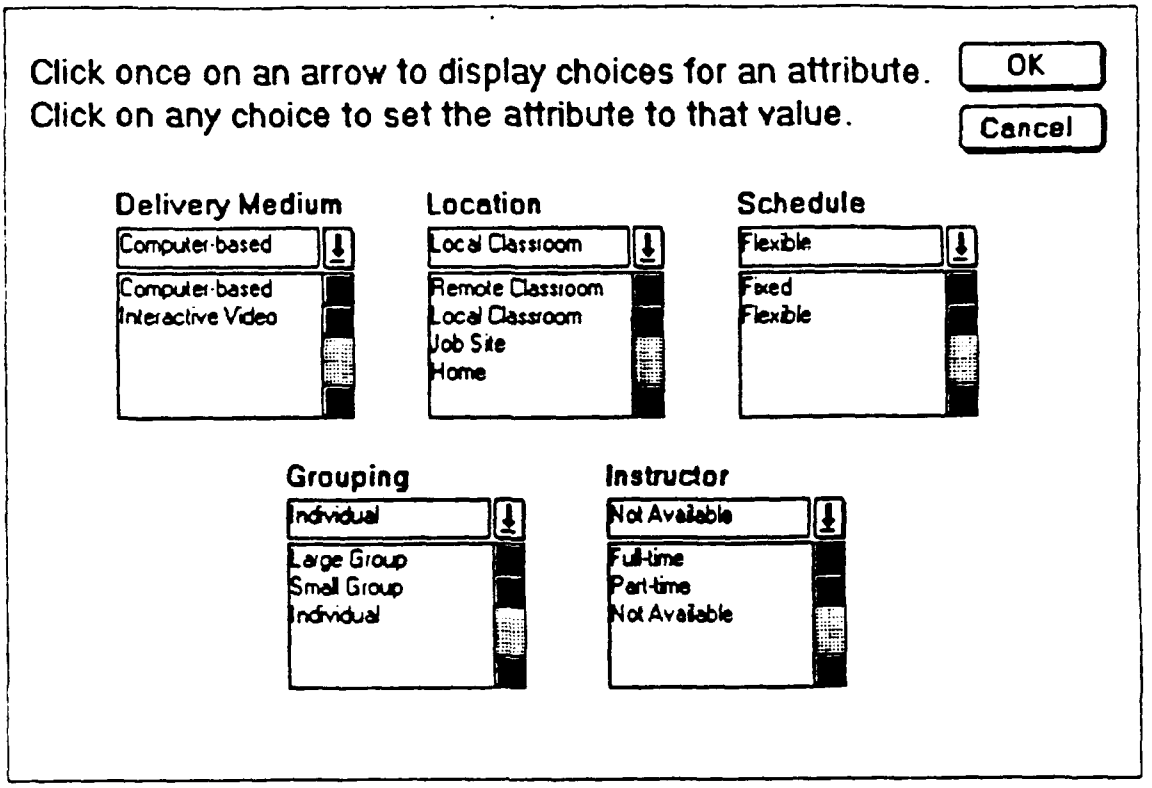

Figure I-6. ID Expert's environment attributes

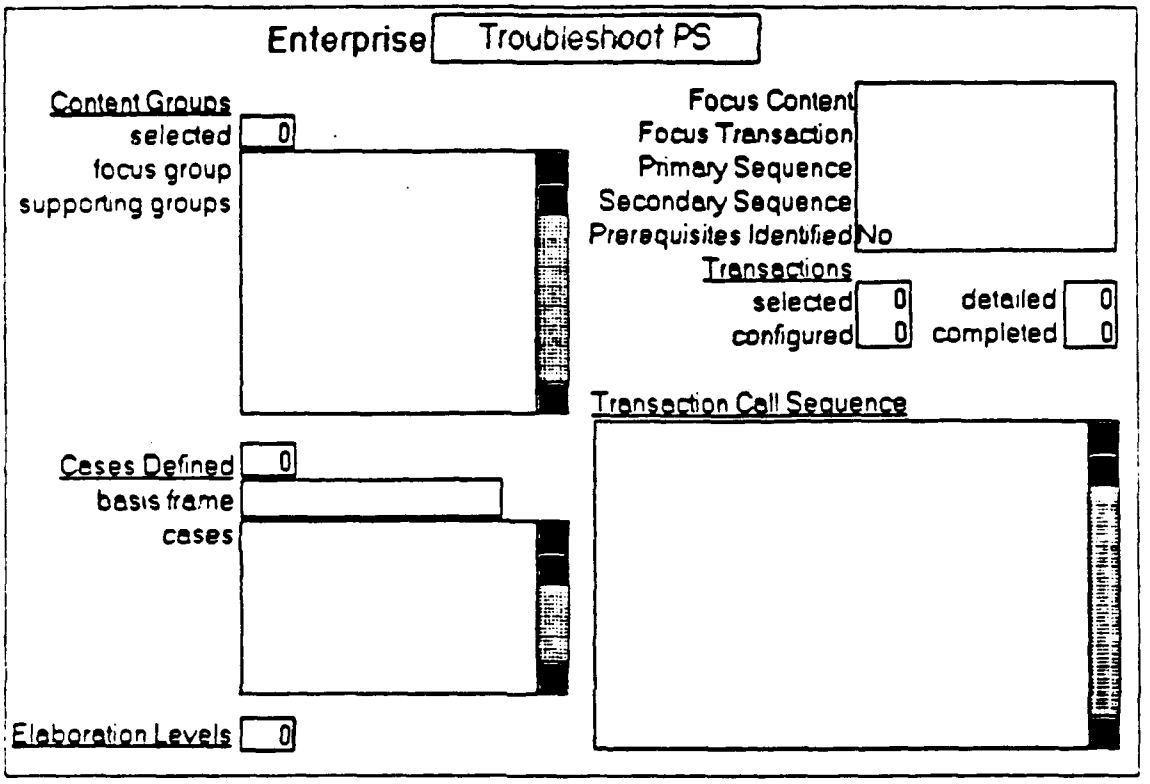

Figure I-7. ID Expert's Enterprise definition

In figure I-5, the user can define attributes of the audience. The environment for the instruction also has a number of attributes that can be set and varied by the user (figure I-6). Before the user can enter the strategy analysis component an enterprise must be defined (figure I-7).

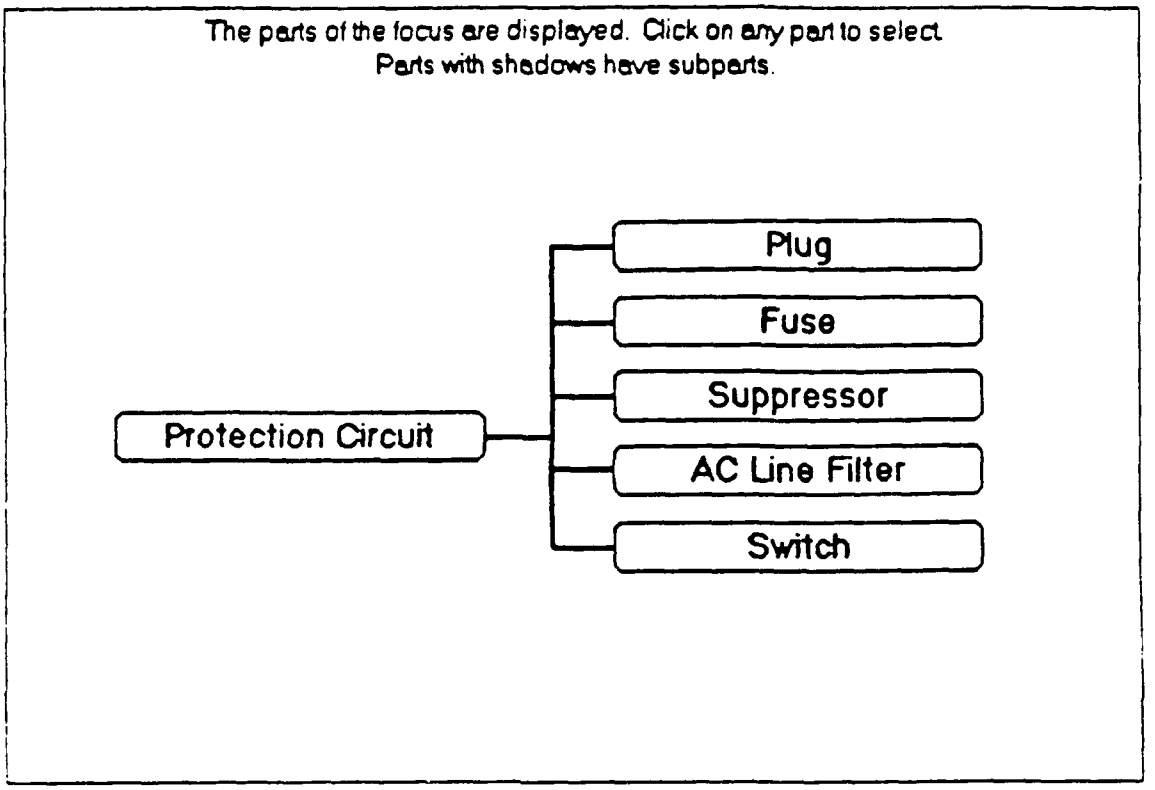

Figure I-8. ID Expert's overview mode

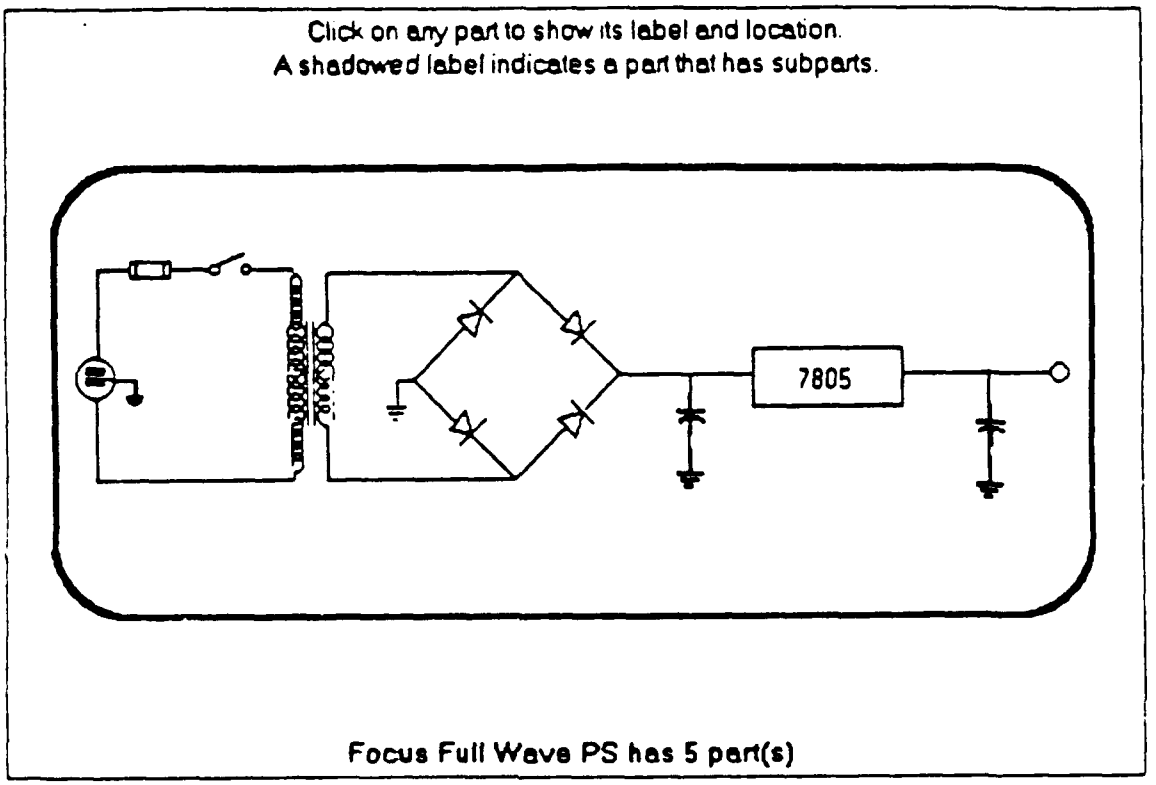

Figure I-9. ID Expert's graphical contents representation

Once the user has made enough decisions, a preview of the transaction can be run through. During this run the user might get into the overview mode of a particular content (figure I-8). Graphical presentation can be used to facilitate the presentation, practice and assessment modes (figure I-9).

4 CONCLUSIONS

In the ID Expert v3.0 report the authors present a list of recommendations for further development of the prototype. A lot of those recommendations are about internal functions and limitations. The most striking recommendations are:

- provide a separate user interface to support the novice user,
- allow multiple inheritance,
- allow constrained inheritance,
- hierarchy depth limitation should be more than the present 5,

- hierarchy breadth limitation (now 8) should be a parameter and settable for a course,
- extend supported properties (now only single valued),
- allow frames to be deleted from the knowledge base,
- provide course verification checks,
- improve existing shells (identify, classify, execute) and provide additional shells (interpret, judge, generalize, propagate, analogize, substitute),
- improve existing enterprises (denote, execute) and provide additional enterprises (manifest, evaluate, design, discover)
- implement delivery adviser,
- extend and use student and environmental characteristics,

Additionally, I have a few other recommendations:

Completeness

It's a good start. However, the system is not complete; there are a lot of elements not implemented or supported. Try to complete the system before extending it.

Interface

The interface looks nice but can be done better. For instance, when you are in the ID Expert Command console it should be directly clear to the user what he has done and what to do next (or what is advised to do next). Maybe for the experienced user this is clear but not for me as a novice user.

Efficient instruction

The delivered instruction might be effective but I am not sure that it is efficient. The produced instruction in the demonstration seems boring and has a lot of identical actions for the student. Try to keep the instruction efficient.

User support

You have to know Merrill's theory to be able to use this job aid efficiently. Although there is some guidance in the system, the user has to be an expert-user to work with the system. Give more user support like online advice, examples, online help and manuals.

5 REFERENCES

Jones, Mark K., M. David Merrill and Zhongmin Li, April 1991. Implementation of an expert system for instructional design. Final report of the Strategy Analysis Component of ID Expert version 3.0. Project report with Human Technology, Inc. and The U.S. Office of Personnel Management.

Merrill, M. David, 1987. The new Component Display Theory: instructional design for courseware authoring. In: Instructional Science, 1987, 16, 19-34.

Merrill, M. David, 1987b. An expert system for instructional design. In: IEEE Expert, Summer 1987, 25-37.

Merrill, M. David, 1988. Applying Component Display Theory to the design of courseware. In: D.H. Jonassen (ed.), Instructional Design for Microcomputer Courseware. Lawrence Erlbaum, Hillsdale, New Jersey, USA.

Merrill, M. David, and Zhongmin Li, 1988. Implementation of an Expert System for Instructional Design (phase 2). Army Research Institute technical report. U.S. Army Research Institute for Behavioral and Social Sciences, Alexandria, Virginia, USA.

Merrill, M. David, and Zhongmin Li, 1989. Implementation of an Expert System for Instructional Design (phase 3). Army Research Institute technical report. U.S. Army Research Institute for Behavioral and Social Sciences, Alexandria, Virginia, USA.

Merrill, M. David and Zhongmin Li, 1989b. An instructional design expert system. In: Journal of Computer-Based Instruction, Summer 1989, Vol. 16, No. 3, 95-101.

Merrill, M. David, Zhongmin Li and Mark K. Jones, January 1990. Limitations of first generation instructional design. In: Educational Technology, January 1990, Vol. 30, No. 1.

Merrill, M. David, Zhongmin Li and Mark K. Jones, February 1990. Second generation instructional design (ID2). In: Educational Technology, February 1990, Vol. 30, No. 2, 7-14.

Merrill, M. David, Zhongmin Li and Mark K. Jones, March 1990. The second generation instructional design research program. In: Educational Technology, March 1990, Vol. 30, No. 3, 26-31.

REPORT DOCUMENTATION PAGE (MOD-NL)

1. DEFENSE REPORT NUMBER (MOD-NL)
TD92-0158

2. RECIPIENT'S ACCESSION NUMBER

3. PERFORMING ORGANIZATION REPORT NUMBER
FEL-92-B059

4. PROJECT/TASK/WORK UNIT NO.
22251

5. CONTRACT NUMBER

6. REPORT DATE
FEBRUARY 1992

7. NUMBER OF PAGES
146 (INCL. 9 APPENDICES
EXCL. RDP + DISTRIBUTION LIST)

8. NUMBER OF REFERENCES
31

9. TYPE OF REPORT AND DATES COVERED
FINAL

10. TITLE AND SUBTITLE
INTELLIGENT TUTORING SYSTEMS; REPORT OF ONE YEAR SABBATICAL LEAVE IN THE USA

11. AUTHOR(S)
J.G.M. VAN DER AREND

12. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
TNO PHYSICS AND ELECTRONICS LABORATORY, P.O. BOX 96864, 2509 JG THE HAGUE
OUDE WAALSDORPERWEG 63, THE HAGUE, THE NETHERLANDS

13. SPONSORING/MONITORING AGENCY NAME(S)
TNO BOARD OF MANAGEMENT
SCHOEMAKERSTRAAT 97, 2628 VK DELFT, THE NETHERLANDS

14. SUPPLEMENTARY NOTES

15. ABSTRACT (MAXIMUM 200 WORDS, 1044 POSITIONS)

THIS REPORT GIVES AN OVERVIEW OF MY STUDY YEAR IN THE USA FROM SEPTEMBER 1990 UNTIL SEPTEMBER 1991. FOR ALMOST FOUR MONTHS, I WORKED ON THE ISTS PROJECT AT THE EMBRY-RIDDLE AERONAUTICAL UNIVERSITY (ERAU) IN DAYTONA BEACH, FLORIDA. THE ISTS (INTELLIGENT SIMULATION TRAINING SYSTEM) IS A TRAINING SYSTEM FOR THE AIR TRAFFIC CONTROLLER USING ARTIFICIAL INTELLIGENCE TECHNIQUES. I HAVE DEVELOPED A LIMITED PROTOTYPE OF THE ISTS ON THE SUN SPARCSTATION USING THE SIMULATION PACKAGE MODSIM II.5.

THE SECOND PART OF THE STUDY YEAR WAS AT THE U.S. ARMY RESEARCH INSTITUTE FOR BEHAVIORAL AND SOCIAL SCIENCES (ARI) IN ALEXANDRIA, VIRGINIA. I WORKED ON THE ANALYSIS AND DESIGN OF ORGANON, A TOOL FOR GRAPHICAL KNOWLEDGE ORGANISATION. ORGANON IS THE FOUNDATION FOR A FUTURE INSTRUCTIONAL DESIGN ENVIRONMENT. I BUILT A PROTOTYPE OF ORGANON ON A NEXT WORKSTATION USING THE OBJECT-ORIENTED PROTOTYPING TOOL INTERFACE BUILDER AND THE PROGRAMMING LANGUAGE C.

THE STUDY YEAR WAS VERY USEFUL AND INSTRUCTIVE. SOME OF THE INFORMATION, KNOWLEDGE AND SKILLS I ALREADY USED IN PROJECTS AT TNO/FEL. THE FIELD OF INSTRUCTIONAL DESIGN IS VERY APPROPRIATE FOR TNO/FEL BECAUSE IN THE FUTURE MANY RESEARCH PROJECTS WILL BE IN THIS FIELD. THE ORGANON CONCEPT CAN ALSO BE USED IN THE FUTURE AUTHORING ENVIRONMENT OF THE FELIX INTELLIGENT TUTORING SYSTEM.

16. DESCRIPTORS
COMPUTER ASSISTED INSTRUCTION
EXPERT SYSTEMS
ARTIFICIAL INTELLIGENCE

IDENTIFIERS
INTELLIGENT COMPUTER ASSISTED INSTRUCTION
INTELLIGENT TUTORING SYSTEMS
INSTRUCTIONAL DESIGN ISTS
ORGANON

17a. SECURITY CLASSIFICATION (OF REPORT)
UNCLASSIFIED

17b. SECURITY CLASSIFICATION (OF PAGE)
UNCLASSIFIED

17c. SECURITY CLASSIFICATION (OF ABSTRACT)
UNCLASSIFIED

18. DISTRIBUTION/AVAILABILITY STATEMENT
UNLIMITED

17d. SECURITY CLASSIFICATION (OF TITLES)
UNCLASSIFIED

Distributielijst

1.	Hoofddirecteur TNO Defensieonderzoek
2.	Directeur Wetenschappelijk Onderzoek en Ontwikkeling
3.	HWO-KL
4. + 5.	HWO-KLu
6.	HWO-KM
7. t/m 9.	Hoofd TDCK
10.	Embry-Riddle Aeronautical University, t.a.v. Dr. A. Kornecki
11.	US Army Research Institute, t.a.v. Dr. R.J. Seidel
12.	Directie IZF-TNO, t.a.v. Dr. ir. A. van Meeteren
13.	Directie FEL-TNO, t.a.v. Ir. P. Spohr
14.	Directie FEL-TNO, t.a.v. Ir. J.W. Maas, daarna reserve
15.	Archief FEL-TNO, in bruikleen aan Ir. J. Bruin
16.	Archief FEL-TNO, in bruikleen aan Ir. M.J. le Mahieu
17.	Archief FEL-TNO, in bruikleen aan Ir. H. Kuiper
18.	Archief FEL-TNO, in bruikleen aan Ir. J.G.M. van der Arend
19.	Documentatie FEL-TNO
20. t/m 29.	Reserves

Indien binnen de krijgsmacht extra exemplaren van dit rapport worden gewenst door personen of instanties die niet op de verzendlijst voorkomen, dan dienen deze aangevraagd te worden bij het betreffende Hoofd Wetenschappelijk Onderzoek of, indien het een K-opdracht betreft, bij de Directeur Wetenschappelijk Onderzoek en Ontwikkeling.